Formalisme pour la conception haut-niveau et détaillée de systèmes de contrôle-commande critiques

Ilias Garnier

UNIVERSITÉ PARIS-SUD

ÉCOLE DOCTORALE INFORMATIQUE DE PARIS-SUD (ED 427)
CEA LIST, Laboratoire des fondements des Systèmes Temps Réel Embarqués

***DISCIPLINE** INFORMATIQUE*

THÈSE DE DOCTORAT
soutenue le 10/02/2012

par

Ilias GARNIER

Formalisme pour la conception haut-niveau et détaillée de systèmes de contrôle-commande critiques

Directeur de thèse :	Guy VIDAL-NAQUET	Professeur (U. Paris-Sud, Supélec)
Co-directeur de thèse :	Christophe Aussagues	Ingénieur de recherche (CEA)
Composition du jury :		
Présidente du jury :	Brigitte ROZOY	Professeur (U. Paris-Sud)
Rapporteurs :	Marc AIGUIER	Professeur (École Centrale Paris)
	Frédéric BONIOL	Professeur (Onera)
Examinateur :	Marc POUZET	Professeur (U. Pierre et Marie Curie)

Résumé

L'importance des systèmes temps-réels embarqués dans les sociétés industrialisées modernes en font un terrain d'application privilégié pour les méthodes formelles. La prépondérance des contraintes temporelles dans les spécifications de ces systèmes motive la mise au point de solutions spécifiques. Cette thèse s'intéresse à une classe de systèmes temps-réels incluant ceux développés avec la chaîne d'outils OASIS, développée au CEA LIST. Nos travaux portent sur la notion de délai de bout-en-bout, que nous proposons de modéliser comme une contrainte temporelle concernant l'influence du flot d'informations des entrées sur celui des sorties. Afin de répondre à la complexité croissante des systèmes temps-réels, nous étudions l'applicabilité de cette notion nouvelle au développement incrémental par raffinement et par composition. Le raffinement est abordé sous l'angle de la conservation de propriétés garantes de la correction du système au cours du processus de développement. Nous délimitons les conditions nécessaires et suffisantes à la conservation du délai de bout-en-bout lors d'un tel processus. De même, nous donnons des conditions suffisantes pour permettre le calcul du délai de bout-en-bout de manière compositionnelle. Combinés, ces résultats permettent d'établir un formalisme permettant la preuve du délai de bout-en-bout lors d'une démarche de développement incrémentale.

Abstract

Real-time embedded systems are at the core of modern industrialized societies. They are a privileged target for the application of formal methods. The importance of real-time constraints in the specification of these systems requires the design of ad-hoc solutions. This work considers a class of real-time systems including those developed using OASIS, a tool-chain targeting hard real-time embedded systems developed at CEA LIST. We study the notion of end-to-end delay, which we propose to model as a constraint bearing directly on the influence of the input information flow over the output information flow . In order to cope with the growing complexity of real-time embedded systems, we study the possibility to apply this new notion of delay to the incremental development of such systems, by using both stepwise refinement and composition operators. We define the necessary and sufficient conditions to the preservation of the end-to-end delay by stepwise refinement. Similarly, we give sufficient conditions to compute the end-to-end delay in a compositional fashion. Together, these results permit to establish a formalism allowing to prove end-to-end delay properties in stepwise development methodologies.

Résumé de la thèse d'Ilias Garnier

L'importance des systèmes temps-réels embarqués dans les sociétés industrialisées modernes en font un terrain d'application privilégié pour les méthodes formelles. La prépondérance des contraintes temporelles dans les spécifications de ces systèmes motive la mise au point de solutions particulières, distinctes des outils logiciels et théoriques traditionnellement utilisées en vérification. Le système OASIS, développé au CEA LIST, propose un socle semi-formel apte au développement de systèmes temps-réels embarqués. Nos contributions sont les suivantes :

- la définition d'un formalisme permettant la vérification de contraintes temporelles de *bout-en-bout*,
- la proposition de solutions pour la vérification de ces contraintes dans le cadre d'une démarche de développement incrémental par raffinement et par composition.

La méthode suivie pour répondre à ces besoins est exposée ci-après. Nous avons montré que les machines de Moore sont un modèle mathématique apte à représenter les systèmes OASIS. Inspiré par les méthodes utilisées dans le champ de la sécurité des systèmes d'information, le délai de bout-en-bout d'un système est directement défini comme le temps de transfert de l'information à travers ce système, permettant ainsi de prendre en compte à la fois les aspects temporels et fonctionnels du délai de bout-en-bout. Cette notion nouvelle a été baptisée le *délai de séparabilité*, dont nous avons plus précisément mis en évidence l'existence de variantes dites *optimistes* et *pessimistes*, selon les hypothèses faites sur l'environnement du système.

Le problème du développement incrémental fut divisé en deux sous-problèmes : le développement incrémental par *raffinement* d'un système donnée, et le développement par composition de sous-systèmes en un système plus complexe. La notion classique de *simulation* a été utilisée pour modéliser le raffinement. Nous avons montré la non-conservation de toutes les variantes du délai de séparabilité par raffinement, suite à quoi nous avons recherché et défini les sous-ensembles de cas les plus généraux dans lesquels il est possible de préserver le délai de séparabilité par raffinement.

Afin de permettre le développement compositionnel, nous avons transposé des travaux existants (portant sur la structure d'algèbre de processus des machines de Mealy) à notre cas. Ceci nous a permis de définir un ensemble minimal et complet d'opérations de composition. La conservation du délai de séparabilité dans le cas de composition séquentielle a alors pu être étudiée en détail. Nous avons montré qu'en général, le délai de séparabilité n'est pas conservé. Nous avons alors défini une propriété plus faible, la réactivité (équivalente au fait d'avoir un délai de séparabilité optimiste fini), et nous avons identifié tous les cas où cette propriété est conservée par composition séquentielle. Cela nous à permis de proposer une classe restreinte de cas où il est possible de donner des résultats de conservation du délai de séparabilité, et donc du délai de bout-en-bout.

En conclusion, nous avons proposé une nouvelle approche pour vérifier une propriété fondamentale de tous les systèmes temps-réels embarqués : le délai de séparabilité. Les propriétés de cette notion par raffinement et par composition ont été étudiées dans le détail, et des solutions aux problèmes rencontrées ont été proposées. Une perspective à court terme pour nos travaux consiste en la vérification de leur applicabilité. A cette fin, il est indispensable d'implémenter un outil de vérification automatique, en mettant à profit certains résultats d'approximation développés plus haut. Une perspective à plus long terme est de procéder à une étude *quantitative* du flot d'information, basée sur la théorie de l'information de Shannon : Dans le cas de systèmes traitant des informations de nature critique, il semble en effet désirable qu'une certaine quantité *au moins* de ces informations soit traitée.

Remerciements. La periode charnière de la fin de thèse est traditionnellement propice à l'introspection. La brusque réalisation du temps passé invite à se remémorer le chemin parcouru, et plus particulièrement les personnes rencontrées.

Je tiens avant tout à remercier Guy Vidal-Naquet, mon directeur de thèse, dont la rigueur patiente m'a permis de mener à bien mes travaux. Merci également à Christophe Aussaguès d'avoir assumé la responsabilité de mon suivi quotidien : les nombreuses discussions que nous avons eues m'ont permis d'éclaircir la jungle foisonnante des idées en les confrontant aux réalités concrètes de mon sujet. Vincent David m'a offert au sein du CEA d'excellentes conditions de travail, ce dont je lui suis reconnaissant. Merci à Frédéric Boniol et Marc Aiguier d'avoir accepté d'être les rapporteurs de ce travail. Ma gratitude est d'autant plus grande qu'il s'agit là d'une charge de travail considérable. Leurs commentaires toujours pertinents me permettront de mieux approfondir mes travaux. De même, merci à Marc Pouzet et à Brigitte Rozoy de m'avoir fait l'honneur de faire partie de mon jury.

Plus loin dans le passé, je tiens à remercier tous les enseignants qui ont participé à ma formation d'informaticien et de chercheur, à commencer par mes professeurs à l'université de Poitiers. Merci en particulier à Patrice Naudin, dont les cours d'algorithmique et de programmation fonctionnelle m'ont permis d'apprécier les aspects formels de ces sujets. Sans la qualité de ces enseignements, je n'aurais probablement pas eu la capacité ni l'envie de partir pour Paris, au MPRI. Ma vision de l'informatique a été profondément changée par ce semestre de découvertes et d'ouverture, et cet esprit m'habite encore aujourd'hui. Mon premier travail de recherche, avec Frédéric Gava, fut un vrai plaisir et constitua la proverbiale *cerise sur le gâteau* – je l'en remercie. Je tiens également à exprimer ma gratitude à deux de mes enseignants au MPRI : Paul-André Melliès et Christine Paulin-Mohring en tant que directrice de l'Ecole Doctorale d'Informatique de l'Université Paris-Sud, pour m'avoir respectivement permis d'assister au séminaire *Game Semantics and Program Verification* à Dagstuhl et pour le soutien matériel de l'Ecole Doctorale. J'ai commencé à développer les travaux contenus dans ce manuscrit quelques semaines après ce voyage, nourri par ce que j'y avais vu et mu par la volonté d'essayer d'explorer moi aussi des voies nouvelles. Enfin, je tiens à adresser une pensée à la mémoire de Zoran Jevtic.

Je me dois également de saluer ceux qui furent mes frères dans l'effort : Nicolas, Thomas, Pablo, Sergiu et Hugo. Merci à tous ceux avec qui j'ai partagé un bureau : Selma, Belgacem, Vincent, Simon, Jean-Thomas et Matthieu. Merci, aussi, à Kods, Emmanuel, Renaud, Stéphane, Thierry, Mathieu, Enzo, Paul, Loïc, Julien, Pascal, Thibaud et Patrick pour votre gentillesse, votre érudition et vos qualités humaines. *Takk* à Tobias, Julien, Alexandra et Simon de m'avoir accompagné dans l'exploration de ce beau pays qu'est l'Islande, et pour d'autres excellents souvenirs. Gloire soit rendue aux intrépides relecteurs de ce manuscrit, dont sont mon père, ma soeur et Simon. Merci à mon vieux frère Thomas, ainsi qu'à tous mes amis de Poitiers. Merci à Friedrich pour l'inspiration, et à Laurent, Maximilien, Damien, Jérôme, Nijel et Johann d'avoir ri avec force de mes exploits et de mes mésaventures au pays des moulins à vent. Beaucoup d'autres personnes mériteraient d'être citées : ne m'en voulez pas trop, vous êtes tellement nombreuses.

Bien qu'il soit imparfait, je dédie à ma mère, mon père et ma soeur ce manuscrit, fruit de trois ans d'efforts.

Table des matières

The whole point of a message is that it should contain something new.

Claude Shannon

1 Introduction

Les sociétés industrialisées sont caractérisées par la remplacement progressif des hommes par des machines dans les tâches de production d'énergie et de biens. L'informatisation a marqué une nouvelle étape dans cette quête de productivité : aux techniciens chargés de la surveillance des processus industriels se substituent graduellement des systèmes semi-automatiques, dits de *contrôle-commande*. Ces systèmes ont de surcroît la particularité d'être en interaction directe avec le monde physique, et d'utiliser une échelle de temps calibrée sur le processus à contrôler. Ils sont donc qualifiés de systèmes *temps-réels*. Du fait des risques écologiques et économiques liés à ces processus industriels, ces logiciels doivent avoir le plus haut degré de fiabilité. Il est généralement admis que cet objectif nécessite des méthodes de conception adaptées.

Nous distinguons plusieurs éléments nécessaires à la constitution d'une telle méthodologie. Le point de départ est toujours la donnée d'une *spécification* du système, ce qui suppose comme préalable d'en avoir un *modèle*. Une des conditions exigée pour la fiabilité est que le logiciel résultant du processus de développement satisfasse la spécification. Outre la garantie de correction que doit apporter la méthodologie, elle doit aider le concepteur à faire face à la complexité de sa tâche en l'aidant à la segmenter. Deux approches classiques peuvent être mises en oeuvre.

- Une segmentation spatiale, par l'assemblage progressif de *composants* modulaires. Cette partition permet d'aider la compréhension, la conception et la vérification du système. Par exemple, chaque sous-système peut se voir assigné un traitement particulier (calcul, communication ...). Dans le cadre des systèmes temps-réel embarqués, cette segmentation sert également l'impératif de fiabilité et de robustesse par la mise en place de mécanismes d'isolation et de tolérance aux fautes. Enfin, elle aide partiellement à prendre avantage des architectures multi-coeurs en découplant les calculs indépendants.
- Un découpage temporel permettant de faire converger graduellement l'objet en cours de développement vers un système fonctionnel par *raffinements successifs* [40, 69].

Nous proposons le *délai de séparabilité*, une notion mêlant contrainte temporelle et fonctionnelle, comme une des principales forces pouvant structurer le développement d'un système temps-réel. Afin de mieux situer cet apport, nous commençons par effectuer un état de l'art recouvrant les points évoqués plus haut.

1.1 Modélisation des systèmes temps-réels

Afin d'exprimer les propriétés attendues d'un système de contrôle-commande, il faut pouvoir en décrire les caractéristiques d'intérêt vis-à-vis de l'environnement dans lequel il est censé fonctionner. Lors d'une démarche purement descriptive, il peut suffire d'énoncer ces propriétés

en langage naturel. Mais lorsque le but est ultimement de prouver la correction du système, il convient d'en avoir un modèle mathématique.

Les systèmes temps-réels sont étroitement liés à la réalité physique. Les modèles les plus précis sont donc obtenus en les plongeant dans le monde du continu, et en décrivant les relations entre leurs entrées, leurs sorties et certaines variables physiques (dont le temps) ; par exemple par la donnée d'équations différentielles. La preuve que le modèle physique exhibe bien les propriétés désirées peut être accomplie formellement. Cette approche permet de surcroît l'utilisation de méthodes de simulation numérique pour prédire le comportement du système voir même d'en générer l'implémentation (par exemple avec Simulink [53]). Cependant, le code généré ainsi n'a à priori aucune garantie formelle de correction.

En effet, il survient nécessairement durant l'implémentation un passage du monde continu au monde discret des calculateurs. Le point crucial est la représentation du temps dans un tel contexte. En général, le modèle de temps discret est obtenu en effectuant des hypothèses simplificatrices sur le modèle physique. Par exemple, un système temps-réel constitué d'entités se déplaçant à des vitesses où la relativité a une influence ne pourrait pas être correctement représenté par une simple horloge commune à toutes les entités. En revanche, cette hypothèse simplificatrice peut être effectuée si le concepteur se contente du cadre de la physique newtonienne. Une autre hypothèse concerne la vitesse de propagation des informations entre les entités constituant le système temps-réel : si ce temps de communication est négligeable par rapport à la dynamique du système physique, le concepteur pourra utiliser un modèle *synchrone*, où une unique horloge discrète commune à toutes les composantes du système régit des communications supposées instantanées. Enfin, des approximations à plusieurs niveaux peuvent être imaginés pour accommoder différentes contraintes, comme les systèmes "globalement asynchrones-localement synchrones".

Ce processus nous mène à traduire la description continue du système dans un modèle formel *discret*. Pour peu que les hypothèses faites soient raisonnables, ce modèle discret peut être considéré comme fidèle à la spécification originelle. Dans nos travaux, nous nous concentrons sur le processus de développement *après* discrétisation.

1.2 Modèles formels discrets

Le choix d'un modèle formel discret dépend bien entendu des hypothèses faites lors du passage du continu au discret. Ce paramètre n'est cependant pas suffisant pour choisir une représentation adaptée à nos besoins. Les aspects suivants doivent être pris en compte lors de ce choix :

- l'aptitude du modèle considéré à représenter un système synchrone ou asynchrone temporisé ;
- les propriétés de compositionnalité et d'expression de la concurrence ;
- l'applicabilité du paradigme de développement par raffinement ;
- l'existence de techniques de vérification automatique pour les propriétés d'intérêt.

Les termes de *systèmes synchrones* et *asynchrones* ont des sens particuliers dans le contexte des modèles formels discrets. Afin de mieux jauger ces modèles, précisons leurs significations respectives.

- Les systèmes synchrones sont caractérisés par une échelle de temps discrète et globale. Les communications entre les composantes du système sont soit immédiates, soit retardées d'un retard borné et connu (afin d'éviter les problèmes de causalité). Un calcul

synchrone est une suite d'étapes calculant la *réaction* du système en fonction de l'entrée courante. Le calcul de la réaction est supposé instantané.
– Les systèmes asynchrones sont constitués de processus évoluant indépendamment les uns des autres, et communicant soit par synchronisations ponctuelles soit par envois de messages. Leurs extensions temporisées les dotent de moyens de mesurer le temps. Ils sont aptes à modéliser des situations où le délai de communication n'est pas négligeable et non-déterministe, comme certains systèmes distribués.

1.2.1 Machines à états et réseaux de Petri

Les machines à états finis sont des outils fondamentaux de l'informatique théoriques. De nombreuses variations en ont été proposées, afin de modéliser les aspects temporels et les calculs réactifs et non-terminants, avec différents degrés d'expressivité.

Cette famille de formalismes présente le défaut de produire des modèles souvent difficiles à manipuler, en particulier lorsque ils atteignent une taille non triviale. Le raffinement d'une machine à état passe généralement par la notion formelle de *simulation*.

1.2.1.1 Automates temporisés

Les automates à états finis classiques n'ont aucune autre notion de temps que celui de la séquence des transitions effectuées. Qui plus est, les modèles familiers étudiés en théorie des langages ne considèrent que des exécutions finies, ce qui semble incompatible avec le comportement d'un système réactif. R. Büchi, dans un autre contexte que l'étude des systèmes réactifs, fut le premier à traiter le cas de séquences d'exécutions infinies [24].

Afin d'exprimer des contraintes temporelles quantitatives, Alur et Dill [7] ont proposé les automates temporisés, une extension des automates de Büchi avec un nombre fini (mais arbitraire) d'horloges en temps continu. Les transitions portent des gardes pouvant faire intervenir ces horloges dans des contraintes de type linéaire simple. Les horloges peuvent également être remises à zéro. L'exécution concurrente de plusieurs processus est modélisée par le produit d'automates. L'ensemble des exécutions possibles du système est alors représenté par tous les entrelacements des exécutions de chaque composante. Le problème de l'inclusion de langage est indécidable dans le cas général, cette relation ne peut donc pas être utilisée comme raffinement, à moins de renoncer à l'automatisation de sa vérification. Il est par contre possible d'utiliser une relation de simulation [25].

D'autres problèmes de vérification sur les automates temporisés sont décidables. La forme restreinte des gardes sur les horloges et le fait que le temps progresse uniformément pour toutes les horloges permet de montrer la décidabilité de l'atteignabilité d'un état, par une approximation finie de l'espace d'états [7]. Les systèmes synchrones sont parfaitement modélisables dans ce cadre [19], mais l'expressivité des automates temporisés est peu mise à profit. Ce formalisme est par contre très bien adapté à la modélisation de systèmes temporisés asynchrones [22]. Des outils permettent de vérifier par model-checking des propriétés de sûreté ou des propriétés encodées dans des logiques temps-réel [60] (voir les outils UPPAAL [58], Kronos [70] et IF [23]).

1.2.1.2 Réseaux de Petri

Les réseaux de Petri sont des modèles calculatoirement plus puissants que les automates à états finis (cf. leur équivalence avec les systèmes d'addition de vecteurs). La modélisation de processus concurrents y est directe. Diverses extensions ont été proposées, y compris des variantes temporisées [61], dont certaines dans l'esprit du modèle de Alur et Dill [63]. En pratique, des restrictions des réseaux de Petri sont utilisées afin d'en restreindre l'espace d'état et de pouvoir appliquer des méthodes de vérification automatique. Les réseaux de Petri dans leur variante temporisée ont été utilisés avec succès pour la modélisation de systèmes asynchrones [61].

1.2.1.3 Machines de Moore et de Mealy

Un modèle de machine à états qui sera d'une importance particulière dans ce manuscrit est celui proposé par Moore en 1956 [56]. Ce modèle est proche des machines de Mealy, proposées à la même époque. Les machines de Moore et de Mealy présentent des caractéristiques communes :

- ce sont des *transducteurs*, associant à tout mot d'entrée un mot de sortie ;
- elles sont originellement déterministes ;
- contrairement aux automates finis "classiques", elles ne permettent pas les transitions invisibles (ϵ-transitions).

Les machines de Mealy se distinguent des machines de Moore par le fait que la valeur de sortie en un instant donné est fonction de leur entrée au même instant ; alors que le formalisme proposé par Moore ne fait dépendre la sortie que de l'état courant.

Depuis leur redécouverte lors de l'avènement du modèle de calcul synchrone, ces types de machines ont été appliqués à la conception compositionnelle de systèmes synchrones, comme en témoigne le langage ARGOS [51]. Ces modèles ont également été étudiés dans le cadre de la vérification automatique de propriétés fonctionnelles et de sûreté par exploration de l'espace d'état (*model-checking*) [28]. L'expression de processus concurrents passe par l'opération de produit synchrone, mais contrairement au cas des automates temporisés, il ne peut y avoir d'entrelacement des exécutions : les composantes du système progressent de façon parfaitement synchrone. Certaines extensions originales des machines de Mealy ont été proposées, en particulier en imposant aux données une structure de treillis [18]. Ceci permet de définir une relation de simulation apte à servir de relation de raffinement.

1.2.2 Langages de programmation formels

Les modèles proposés précédemment sont aptes à modéliser les systèmes synchrones ou asynchrones, mais restent relativement bas-niveau. Ils ne peuvent en particulier pas être utilisés lors de la phase d'implémentation proprement dite. Afin de pouvoir produire une implémentation *correcte*, il est indispensable que le langage de programmation repose lui-même sur un socle formel. Tous les langages présentés ici permettent l'expression de processus concurrents. Les techniques de vérification reposent en général sur la génération d'une machine à états correspondant au programme analysé, via la sémantique opérationnelle du langage utilisé.

1.2.2.1 Langages synchrones

Approche réactive. Esterel [14] est un langage synchrone impératif largement utilisé pour la programmation de systèmes temps-réels. Il n'est pas un langage réactif *stricto sensu*, du fait des problèmes de causalité induits par ses primitives de gestion de signaux. Il a en cela motivé la naissance d'un modèle de programmation libéré de cet inconvénient. Le langage Reactive C [20] étend le langage C avec un ensemble de primitives réactives orthogonales au langage hôte et n'induisant pas de boucle de causalité. Cependant, son langage hôte fait que Reactive C n'est pas strictement synchrone. Ce modèle a ensuite donné naissance au langage SL [21]. Plus récemment a été proposé ReactiveML [50], une extension réactive du langage fonctionnel OCaml.

Approche flots de données. Lors de la discrétisation du modèle physique du système temps-réel, les variables dépendantes du temps sont transposées en séquences infinies d'échantillons, et les relations (i.e. les équations différentielles) entre ces variables sont implémentables comme des fonctions *causales* sur ces flots de données. Les langages Lustre [38] et Signal [8] permettent d'implémenter de telles fonctions.

Approche OASIS. La chaîne OASIS [32] comprend le langage de programmation PsyC. Un programme PsyC est constitué d'un ensemble statique d'agents communiquant soit par envoi de messages, soit par flot de donnée. Une instruction particulière permet à chaque agent de préciser explicitement les instants formels auxquels son état observable est mis à jour. Du fait de son utilisation de C, le langage PsyC doit être classé parmi la famille des langages semi-formels. L'approche OASIS constitue le point de départ de nos travaux.

1.2.2.2 Algèbres de processus

Les algèbres de processus tiennent leurs noms du fait que leurs syntaxes incluent des opérateurs de composition (séquentielle, parallèle, ...). Des formalismes comme CCS [54] et CSP [41] ont été conçus dans le but de modéliser et d'étudier les aspects *concurrents* dans les systèmes informatiques, et en particulier les synchronisations entre tâches. Les premiers modèles ne disposaient pas de notion de *temps* autre que l'ordre relatif imposé par les synchronisations. Par la suite, plusieurs extensions temporisées ont vu le jour.

Algèbres synchrones. Une extension synchrone de CCS nommée SCCS a été proposée par Milner en 1983 [55]. Austry et Boudol proposent la même année une vision complémentaire (et expressivement équivalente) à SCCS avec l'algèbre Meije [10], dont la composition parallèle est asynchrone, mais qui dispose d'opérations de synchronisation. Ces deux derniers modèles sont dédiés à l'étude de leurs propriétés intrinsèques. L'algèbre CoReA, proposée dans [17], est tournée vers la modélisation des systèmes synchrones *distribués*. L'auteur observe qu'un temps de communication nul est inapte à modéliser les communications entre agents distribués. L'auteur propose de se reposer sur un modèle synchrone *faible*, où les communications prennent un délai d'exactement une unité de temps. Cette algèbre peut alors servir de couche de coordination pour des agents programmés dans un langage synchrone fort.

La grande variété d'algèbres de processus a motivé la recherche d'un modèle libéré des variations syntaxiques, et inspiré des techniques de sémantique dénotationnelle [2]. Abramsky

adopte en 1996 une approche algébrique, exprimée dans le cadre de la théorie des catégories, qui cherche à *déduire* un formalisme à partir des qualités observables du système. Cet auteur propose en particulier une algèbre de processus synchrone nommée SPROC, suffisamment expressive pour encoder les langages synchrones Lustre, Esterel et Signal [3].

Algèbres asynchrones. D'autres auteurs ont proposé d'étendre les algèbres de processus non-temporisées avec des constructions permettant de spécifier des contraintes quantitatives similaires à celles trouvées dans les automates temporisés. Des représentants de cette famille sont CCS rt et Timed CSP [57].

Propriétés des algèbres de processus. Ces langages sont pour la plupart complets au sens de Turing et ne se prêtent pas bien à la vérification automatique. Les algèbres de processus ont permis d'inspirer de nombreux autres formalismes pour l'expression de la composition et les mécanismes de synchronisation ainsi que par l'étude précise des propriétés de congruence de différentes notions d'équivalences comportementales. Comme dans le cas des machines à états, le raffinement peut être abordé par l'utilisation des pré-ordres de simulation.

1.3 Spécification

Lors de la phase de conception d'un système de contrôle-commande, la spécification et le modèle continu sont traduits et décomposés manuellement en un ensemble de contraintes de nature diverses. L'exigence la plus importante dans le champ des systèmes de contrôle-commande est que l'asservissement du processus physique réalisé par ledit système soit correct. En d'autres termes, le système doit calculer *ce qu'il faut*, et *en temps voulu*. Cette dichotomie introduit dans le domaine discret les deux classes de spécifications *fonctionnelles* et *temporelles*.

1.3.1 Spécifications fonctionnelles

La vérification de la correction des programmes n'est pas spécifique à l'informatique embarquée. Pour la classe des programmes transformationnels et non-réactifs, il s'agit de s'assurer qu'un programme calcule bien la fonction désirée (au sens mathématique). Il s'agira par exemple de la correction d'un algorithme de tri ou d'une passe de compilation.

Un programme réactif est constitué d'une suite potentiellement infinie de calculs finis. Cependant, l'étude de spécifications existantes (par exemple le cas d'étude du *steam-boiler* d'Abrial [4]) montre que les propriétés fonctionnelles y étant exprimées sont souvent de nature plus simple que dans le cas transformationnel, comme le fait qu'une variable donnée reste dans un domaine particulier.

1.3.2 Spécifications temporelles

Nous distinguons parmi les spécifications temporelles celles d'ordre quantitatif de celles d'ordre qualitatif.

Contraintes quantitatives. Afin que le processus physique contrôlé par le système temps-réel se déroule normalement, le système doit pouvoir *observer* et *réagir* en un temps qui soit assez court pour permettre le maintien de ce processus sous contrôle. Cela induit des contraintes *quantitatives* sur l'intervalle temporel séparant les deux états du système correspondant à l'observation et à la réaction associée. La définition effective de ces états est du ressort du concepteur.

La durée effective de ces temps dépend de l'application : dans le cas du contrôle d'une réaction chimique lente, il sera superflu d'activer le système à chaque milliseconde. Ce sera au contraire primordial lors du contrôle-commande d'une fusée. Dans tous les cas, le respect scrupuleux de ces contraintes est constitutif de la correction globale du système. Dans le cas de systèmes constitués de plusieurs sous-systèmes communicants, la contrainte temporelle est teintée d'une composante *spatiale*. Une donnée en entrée du système peut en effet être traitée successivement par plusieurs tâches fonctionnant en parallèle avant d'être disponible en sortie. Nous parlerons alors de *contraintes de bout-en-bout*.

Contraintes qualitatives. L'autre type de spécification temporelle, moins spécifique au temps-réel, impose des contraintes de précédence sur certains états du système. Par exemple, certains logiciels embarqués effectuent un auto-test *avant* de procéder au démarrage effectif du système de contrôle-commande. Ce type de contrainte est particulièrement prégnant lors de la spécification de protocoles de communication, et ne sera pas abordé plus avant.

1.4 Conception incrémentale

Le développement incrémental de systèmes temps-réels a motivé la conception de nombreux ensembles d'outils.

1.4.1 Conception incrémentale par raffinement

Avant d'explorer l'état de l'art en la matière, nous rappelons les quelques principes qui sous-tendent ces méthodes. Nous pouvons distinguer cinq éléments clefs : le langage de spécification, le langage d'implémentation, la relation de satisfaction entre spécification et implémentation, la relation de raffinement de spécification, et enfin la relation de raffinement d'implémentation.

Le langage de spécification permet d'édicter le comportement du système ou du sous-système. Le langage d'implémentation propose comme son nom l'indique une représentation *concrète*, voire exécutable du système. La relation de satisfaction s'assure qu'une implémentation se comporte comme précisé dans sa spécification. Enfin, les relations de raffinement permettent de rajouter des détails et de compléter les parties manquantes du système.

Afin de constituer une méthode de développement utile, certaines propriétés de cohérence doivent être vérifiées par ce quintuplet. Si une implémentation I satisfait une spécification S, tout raffinement I' de l'implémentation doit satisfaire cette spécification S. De façon covariante, l'implémentation I doit satisfaire toute spécification S' telle que S est un raffinement de S'. De plus, le raffinement doit être compositionnel : une modification locale doit avoir un impact local.

Plusieurs formalismes ont été proposés à cet effet. Un des plus expressifs est Event-B, qui est basé sur le formalisme de la théorie des ensembles. L'implémentation est une machine à états

non-déterministe, et le raffinement est une forme de simulation. Un programme C ou ADA est automatiquement généré lorsque l'implémentation devient (par raffinement) déterministe.

Une extension notable du modèle d'Alur et Dill [7] sont les *Timed I/O automata* [45, 31]. Ce formalisme a été créé afin de modéliser la composition et le raffinement de systèmes temporisés. Il propose de surcroît une opération de *quotient* permettant de déduire automatiquement la spécification la plus générale dans une spécification globale lacunaire. Les *Timed Interfaces* [6] présentent la particularité de proposer une notion de satisfiabilité optimiste : une implémentation est correcte si *il existe* un contexte dans lequel elle respecte sa spécification.

1.4.2 Conception incrémentale par composition

Le formalisme BIP [11] permet la modélisation de systèmes hétérogènes (par exemple, de systèmes GALS) et dispose d'un outil de génération automatique de code. Dans le cas asynchrone, la détection automatique d'inter-blocages peut être effectuée à l'aide de l'outil D-Finder [12]. Une des perspectives de développement de BIP consiste à prouver formellement la correction de la génération de code et à interfacer le générateur avec un compilateur certifié.

La suite d'outils SCADE permet le développement de systèmes synchrones par l'utilisation d'un modèle flot de donnée reposant sur le langage Lustre, et par la conception hiérarchique de machines à états. Elle inclut entre autres un simulateur d'exécution, un analyseur WCET et un outil de model-checking afin de vérifier aussi bien les propriétés temporelles que fonctionnelles [1]. L'outil Syndex est dédié au *co-design* de systèmes temps-réel embarqués sur machines parallèles. Le programme y est représenté par un graphe représentant les flots de données entre des processus [66].

1.5 Plates-formes logicielles pour le temps-réel

Une fois le développement achevé, l'exécution correcte du système discrétisé repose sur l'accès aux ressources de la machine hôte, à commencer par une horloge matérielle. Cet accès peut être direct ou passer par un support d'exécution logiciel. Le choix de ce support d'exécution influe fondamentalement sur le degré de sûreté du système pris dans son ensemble. Notre support d'exécution est fourni par le système d'exploitation OASIS.

1.5.1 Plates-formes pour les systèmes asynchrones

L'approche dominant encore aujourd'hui le développement de systèmes temps-réels embarqués repose sur l'utilisation de systèmes d'exploitation spécialement conçus pour permettre l'exécution de tâches temps-réel. Quelques exemples de tels systèmes sont VxWorks, Linux + RTAI et QNX. Les tâches elles-mêmes sont programmées dans des langages comme C ou ADA, en utilisant les primitives fournies par le système sous-jacent pour implémenter les contraintes temporelles édictées par la spécification. Le système d'exploitation s'assure que les primitives appelées par la tâche sont bien exécutées et la coordination entre tâches est effectuée à l'aide de primitives de synchronisation généralistes, comme les sémaphores.

Dans le cas où le système d'exploitation permet l'exécution parallèle de tâches (comme Linux ou VxWorks), cette approche permet la conception de systèmes pouvant tirer parti des nouvelles architectures multi-coeurs et ainsi fournir la puissance de calcul requise par les applications. Néanmoins, l'utilisation de langages de programmation tels que C fait apparaître le

risque d'erreurs dans les manipulations de pointeurs, de dépassement arithmétique, de dépassement de tableaux, de mauvaise gestion des ressources ... De plus, l'utilisation de primitives de synchronisation et de communication trop générales introduit la possibilité de comportements incorrects. Un exemple classique est celui des inter-blocages induits par une mauvaise utilisation des sémaphores. Ces problèmes ont motivé le développement des modèles formels et des techniques de vérification présentés ici.

1.5.2 Plates-formes pour les systèmes synchrones

La principale technique de compilation pour les langages synchrones consiste en la génération d'un automate fini. Il est ensuite du ressort du concepteur de calibrer la durée d'un pas de l'horloge synchrone sur le processus physique asservi. Plusieurs techniques d'exécution sont envisageables :

- la génération directe de circuits logiques matériels implémentant la machine à état,
- l'exécution directe sur une cible matérielle standard,
- l'exécution en mode utilisateur sur un système d'exploitation temps-réel.

Cependant, un programme synchrone compilé en un unique automate est difficilement exécutable sur plusieurs coeurs de calcul. Certains travaux récents cherchent à permettre l'exploitation de ces ressources de calcul [26].

1.5.3 Approches mixtes

Une partie de ces problèmes est résolue par l'utilisation d'outils particuliers, comme la suite d'outils OASIS qui permet la mise au point de systèmes temps-réel embarqués pouvant prendre avantage des architectures multi-coeurs. OASIS permet également la vérification durant l'exécution du respect de certaines contraintes temporelles par l'application embarquée, grâce à son socle formel synchrone. Les tâches sont programmées dans un langage dédié étendant C avec des instructions de synchronisation par le temps, et des primitives de communication par flots de données et par messages. L'exécution d'un programme OASIS est déterministe dans le temps. Cette approche permet donc de profiter des architectures multi-coeurs sans risque d'inter-blocage. Ceci ne répond pas, néanmoins, au besoin d'une méthode de développement spécifique au domaine ni à la vérification avant exécution du respect de la spécification par le système.

1.6 Travaux connexes

Nos travaux sont fortement inspirés des analyses sur la sécurité du flot de l'information. Ces approches considèrent principalement les systèmes transformationnels et ont pour but de s'assurer que des données privées (comme un mot de passe) au sein d'un programme ne sont pas inférables par une attaque extérieure. La propriété correspondante est nommée *non-interférence* [37] et repose sur l'étude de dépendances fonctionnelles entres les données privées et un ensemble d'observables modélisant le pouvoir de l'attaquant. Le cas réactif a été considéré entre autres par [16] dans le contexte de la sécurité des navigateurs web. Dans [9], une analyse du flot d'information a été entreprise dans le cadre d'une variante synchrone de CCS. Ces travaux se rapprochent des notres en ce qu'ils considèrent des systèmes non-déterministes, et se basent sur la notion de bisimulation pour définir la non-interférence. Nous procédons à une démarche

duale : nous cherchons à construire des systèmes réactifs synchrones en garantissant l'existence d'un flot d'information entre les entrées et les sorties.

Dans son article fondateur *Gedanken Experiments on Sequential Machines* [56], E. F. Moore expose un moyen de *distinguer* deux états l'un de l'autre par le biais d'expériences. Cette notion d'états distinguables est très proche de notre notion de séparabilité. Le délai de séparabilité sera intuitivement lié à la longueur de l'expérience permettant de prouver que deux états sont distinguables. Cette approche porte aujourd'hui le nom de *test d'automates* [47, 67]. Son objectif général est de formaliser les moyens par lesquels il est possible d'extraire de l'information depuis une machine en observant sa réaction à certaines séquences d'entrées.

Les machines de Moore et de Mealy définissent des fonctions *causales* sur les flots de données. Cette notion de causalité exprime le fait que l'état observable d'une machine au temps t n'est fonction que des entrées aux temps $[0; t]$ et de l'état initial. La génération automatique de fonction causales sous la forme de machines à états à partir de spécifications logiques a fait l'objet de quelques travaux (voir par exemple [39]). La notion que nous proposons peut être considérée comme un enrichissement de la causalité, où le temps précis entre une donnée en entrée et sa conséquence observable est spécifié.

Les circuits digitaux ont fourni un modèle d'étude de la sémantique des langages synchrones (pour Esterel, voir par exemple l'état de l'art de [64])). Or, les circuits sont également soumis à des contraintes temporelles fortes. En particulier, l'analyse du délai de bout-en-bout induit par un circuit est largement pratiquée sous l'appellation d'*analyse temporelle statique* [15]. Le délai total est alors calculé comme le nombre maximums de délais logiques sur tous les chemins entre les entrées et les sorties. Par vertu de l'équivalence entre programmes synchrones et circuits digitaux, il est en principe possible d'appliquer cette analyse à des programmes.

Une approche pragmatique à l'analyse du délai de bout-en-bout dans les systèmes temps-réels multi-tâches a été proposée dans la thèse de Tanguy Le Berre [46]. Ce dernier propose de modéliser une tâche par les dates de mise à jour de ses variables (correspondantes aux flots de données en entrée et en sortie). Ceci permet de prendre en compte des tâches non périodiques. Il est ainsi possible de calculer l'intervalle temporel minimal entre une entrée en un point du système et une sortie dans une autre tâche, en prenant en compte les communications entre tâches. L'auteur propose une méthode pour analyser ce délai de façon automatique. Nous nous distinguons de ces travaux par le fait que nous prouvons une dépendance fonctionnelle effective entre une entrée et une sortie.

Le terme de *séparabilité* est également utilisé dans d'autres domaines, pour désigner certains types d'espaces topologiques ainsi que dans l'étude des systèmes d'addition de vecteur. Ces utilisations sont sans rapport avec les développements contenus dans ce manuscrit.

1.7 Contribution proposée

L'importance capitale de la correction du comportement temporel des systèmes temps-réel embarqués motive un examen critique des solutions existantes au problème de la spécification et de la vérification des contraintes de bout-en-bout. Nous avons vu que ces contraintes quantitatives sont équivalentes à la donnée d'un intervalle temporel entre d'une part le passage dans un état correspondant à la lecture d'une donnée et d'autre part l'*observation d'une réponse*.

Dans les approches classiques, qu'il s'agisse de systèmes synchrones ou asynchrones, cette notion de réponse observable est réduite à l'occurrence d'un évènement la *symbolisant*, par exemple par la mise à jour d'une variable. Dans tous les cas, il s'agit d'une approximation d'un

processus *étendu dans le temps* de transformation de données entre les entrées et les sorties du système. Nous avançons qu'il est possible de se passer de cette approximation.

En conséquence, nous proposons de substituer à la notion classique de contrainte temporelle une contrainte mêlant aspects fonctionnels et temporels : l'existence d'un flot d'information entre les entrées et les sorties du système. Nous appelons cette nouvelle contrainte le *délai de séparabilité*.

Modèle formel. Nos développements ont pour cadre les systèmes OASIS, pour lesquels nous déterminons comme modèle formel une variante des machines de Moore. Notre modèle se distingue de celui d'origine par les aspects suivants :

- afin de modéliser les systèmes en cours de développement, nos machines ne sont pas déterministe ;
- qui plus est, elles disposent de multiples ports d'entrées et de sorties.

Nous montrons la validité du choix des machines de Moore comme modèle en définissant une sémantique opérationnelle à grand pas pour une restriction raisonnable du langage PsyC (dans le cas déterministe).

Spécification. La notion de séparabilité est définie sur ces machines comme une adaptation du concept de *dépendance fonctionnelle*. Ces dépendances fonctionnelles reposent dans notre cas sur une notion de preuves de non-bisimilarité, qui nous permet de décliner le délai de séparabilité en une variante pessimiste et une variante optimiste, selon que l'on souhaite *garantir* ou savoir *possible* l'occurrence d'une réponse observable à une entrée particulière. Nous dédions le reste du manuscrit à l'étude des propriétés du délai de séparabilité dans le cadre d'une méthodologie de développement par raffinement et par composition.

Raffinement. Nous faisons le choix classique de la relation de simulation comme relation de raffinement. Des conditions suffisantes à la conservation du délai de séparabilité sont données pour cette relation. Un formalisme de développement se basant sur ces résultats est alors proposé, où les données peuvent également être raffinées. Afin de permettre la vérification efficace du délai de séparabilité entre des ports d'entrées et de sorties précis, nous étudions une technique d'abstraction de machine et nous définissons les conditions sous lesquelles elle est applicable.

Composition. Afin d'étudier le délai de séparabilité sous composition, nous adoptons une approche minimaliste inspirée des algèbres de processus, et plus particulièrement de certains travaux de Samson Abramsky [3]. Une attention plus particulière est apportée à la composition séquentielle de machines de Moore. Comme pour le raffinement des conditions suffisantes pour garantir la conservation du délai de séparabilité sont données. Combinés, ces résultats forment une nouvelle approche pour l'étude et le développement des contraintes de bout-en-bout.

1.8 Plan du manuscrit

Le chapitre 2 (p. 23) est consacré à la définition de notre cadre d'étude, constitué par une variante non-déterministe des machines de Moore. Le pré-ordre de simulation sur les machines

de Moore est défini, et le délai de séparabilité est introduit. Les propriétés de conservation du délai de séparabilité sont étudiées.

L'applicabilité des techniques d'abstraction est étudiée dans le chapitre 3 (p. 51). Quelques résultats de conservation du délai de séparabilité sont prouvés, ainsi que des résultats sur la possibilité d'analyser le délai de séparabilité dans le cas ou la machine dispose de plusieurs ports d'entrée et de sortie. Ces résultats sont appliqués à la proposition d'un formalisme permettant le raffinement de machines préservant le délai de séparabilité.

Les propriétés de compositionnalité du délai de séparabilité sont étudiées dans le chapitre 4 (p. 93). Après avoir défini quelques opérations de composition, la composition séquentielle de machines est montrée comme ne préservant pas le délai de séparabilité, et des conditions suffisantes à sa préservation sont démontrées. Une approximation compositionnelle du délai de séparabilité en est déduite.

Nous effectuons une revue critique de nos travaux et présentons des perspectives de recherche en conclusion (Chapitre 5 p. 115).

En annexe (p. 121), nous établissons un lien formel entre nos travaux et la plate-forme OASIS par la définition d'une sémantique opérationnelle pour le langage PsyALGOL, inspiré de PsyC.

2

Machines de Moore et délai de séparabilité

Ce chapitre a pour objet une définition du délai de séparabilité d'un état d'une machine de Moore. Nous commençons par poser des définitions générales dont celle de machine de Moore. Nous poursuivons par la définition dans ce modèle formel d'une notion de délai de séparabilité dont nous exposons diverses propriétés.

2.1 Préliminaires

Nous rappelons quelques définitions mathématiques liées à la théorie des langages formels.

2.1.1 Définitions mathématiques

2.1.1.1 Définitions générales

1. L'ensemble singleton est noté $\mathbb{1} = \{\star\}$ (à isomorphisme près).
2. La relation identité est notée Id. Si $R_1 \subseteq A \times B$ et $R_2 \subseteq B \times C$, la composition relationnelle est notée $R_1; R_2 \subseteq A \times C$.
3. Si $f \in A \to B$ et $g \in B \to C$ sont deux fonctions, leur composition est notée $g \circ f \in A \to C$.
4. Si $f \in A \to B$ et $g \in C \to D$, l'extension du produit cartésien $\times$ aux fonctions est notée $\langle f, g \rangle \in A \times C \to B \times D : \langle f, g \rangle = (x, y) \mapsto (f(x), g(y))$.
5. Soit $A \times B$ le produit cartésien de A et B. Les deux fonctions de projection associées seront notées :
$$\pi_A : A \times B \to A,$$
$$\pi_B : A \times B \to B.$$

2.1.1.2 Alphabets et langages

Soit un ensemble Σ. Nous pouvons construire à partir de Σ deux ensembles particuliers : celui des mots finis Σ^* et celui des mots infinis Σ^ω. Dans les deux cas, Σ est appelé *alphabet*.

1. Σ^* est le monoïde libre, c'est-à-dire l'ensemble des séquences *finies* d'éléments de Σ. L'élément neutre de Σ^* est le mot vide, noté ϵ.
2. La longueur d'un mot w est notée $|w|$, avec $|\epsilon| = 0$.
3. Σ^+ correspond aux mots finis de longueur non-nulle.

4. Σ^ω est l'ensemble des séquences *infinies* d'éléments de Σ. Σ^ω est défini formellement comme $\mathbb{N} \to \Sigma$, c'est-à-dire comme l'ensemble des fonctions associant à tout entier naturel (i.e. l'indice dans la suite) un symbole de Σ.
5. Nous définissons l'ensemble des mots finis et infinis comme $\Sigma^\infty = \Sigma^* \bigcup \Sigma^\omega$.
6. Si $w \in \Sigma^\infty$ est un mot, nous notons son i-ème symbole $w[i]$, où i varie entre 0 et $|w| - 1$.
7. Si $w \in \sigma^\infty$ est un mot, l'ensemble de ses préfixes finis est noté $\mathsf{prefix}(w)$. Cet ensemble contient w si et seulement si w est fini.
8. Si $L \subseteq \Sigma^*$ est un ensemble de mots, la partie de L constituée des mots de taille *minimale* est notée $\mathsf{Minimal}(L)$:

$$\mathsf{Minimal}(L) \;=\; \{w \;\in\; L \mid \forall w' \in L, |w| \leq |w'|\}\,.$$

2.1.2 Types de données : le monoïde $\mathfrak{D}$

Nous supposons la préexistence de quelques types de données basiques, au premier rang desquels figurent les entiers finis **int** ainsi que les booléens **bool** :

$$\begin{array}{lcl} \textbf{int} & = & \{min\_int; \ldots; max\_int\}, \\ \textbf{bool} & = & \{\textbf{tt}\,; \textbf{ff}\,\}. \end{array}$$

Soit $Basic = \{\textbf{int}; \textbf{bool}; \ldots\}$ cet ensemble de types de base. L'ensemble des types de données est le monoïde $Type = \langle \mathfrak{D}, \times, \mathbb{1} \rangle$ généré par $Basic$ et fermé par produit cartésien. L'élément neutre de ce monoïde est $\mathbb{1} = \{\star\}$ (à isomorphisme près). Nous travaillerons également avec les isomorphismes correspondants à la commutativité $(a, b) \mapsto (b, a)$, à l'associativité $(a, (b, c)) \mapsto ((a, b), c)$ et à l'absorption de l'élément neutre $(a, \star) \mapsto a$.

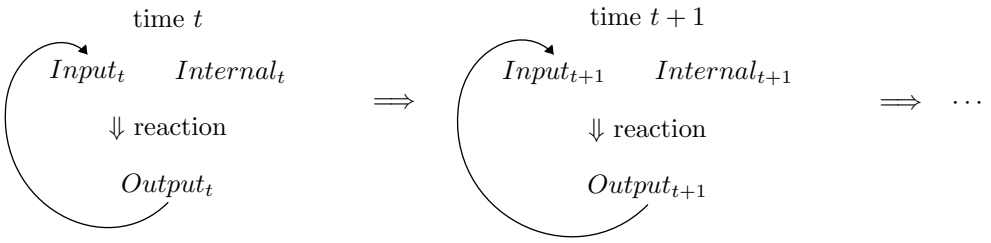
Les éléments de $\mathfrak{D}$ serviront d'alphabets d'entrée et de sortie dans la définition de notre modèle synchrone. Les isomorphismes cités plus haut pourront être étendus aux éléments du modèle que nous allons définir et serviront alors à restructurer les interfaces des machines (voir chapitre 3, Sec. 3.3 p. 72).

2.2 Modèle formel de systèmes synchrones

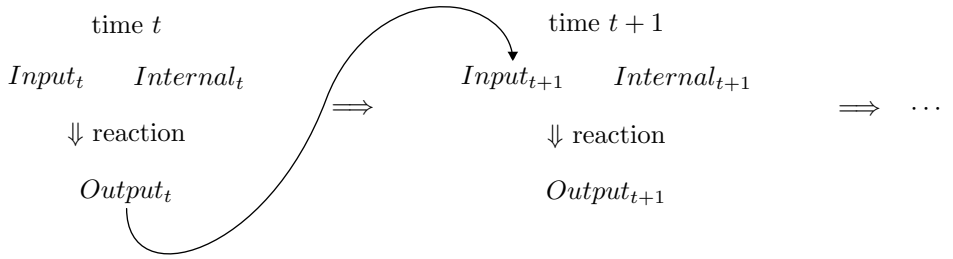
Notre propos est ici de donner un modèle des systèmes synchrones qui soit suffisamment expressif pour encoder le modèle proposé par OASIS [32] (cf. Annexe A). Après avoir donné une définition informelle du comportement des systèmes à modéliser, nous en donnerons une description sous forme de machine à états.

2.2.1 L'abstraction synchrone faible

Le modèle synchrone repose sur une notion d'horloge logique permettant d'orchestrer les calculs. A chaque temps logique t, chaque composante du système calcule à partir de ses entrées et de son état interne une réaction et un nouvel état interne. Dans le modèle synchrone fort, cette réaction est directement visible au temps t, ce qui peut causer des problèmes de causalité (voir le cas d'Esterel dans [13]). La figure ci-dessous illustre le modèle synchrone fort par deux étapes de calcul d'un "processus" synchrone isolé.



Une première solution à la non-causalité est d'exclure les programmes non-causaux, par exemple par une analyse statique du code. Une autre façon de procéder est d'affaiblir le modèle en ne rendant visible les réactions du temps t qu'au temps $t + 1$. Ceci interdit par construction les dépendances cycliques entre entrées et sorties. Ce modèle synchrone est dit *faible* et est utilisé dans le langage SL [21] et ses descendants, comme Reactive ML [50]. Il est également au coeur de l'algèbre de processus CoReA [17] et du langage PsyC inclus dans OASIS, comme le montre l'étude de sa sémantique opérationnelle (cf. Annexe A). Ce modèle est illustré par la figure suivante.



2.2.2 Description de systèmes faiblement synchrones comme machines à états

Nos travaux portent sur une variété particulière de transducteurs synchrones permettant d'implémenter le modèle synchrone faible. Nous définissons nos systèmes synchrones comme des machines à états connues sous le nom de *machines de Moore* [56].

Définition 1 (Machine de Moore)**.** *Soit In un alphabet* d'entrée *et Out un alphabet de* sortie*. Une machine de Moore M sur In et Out est la donnée de :*

- *un ensemble fini d'états noté Q,*
- *un ensemble d'arcs $E \subseteq Q \times In \times Q$ tel que pour tout état q et toute donnée $in \in In$, l'ensemble des arcs $\{(q, in, q') \in E \mid q' \in Q\}$ est non-vide (condition de* totalité*),*
- *une fonction* $\mathsf{out} : Q \to Out$ *associant à tout état un symbole de sortie,*
- *un* ensemble *d'états initiaux $Q_i \subseteq Q$.*

Une telle machine est notée $M = \langle In, Out, Q, \mathsf{out}, E, Q_i\rangle$. Si $Q_i = \{q_i\}$, nous le noterons directement par $\langle In, Out, Q, \mathsf{out}, E, q_i\rangle$. Nous noterons $p \xrightarrow{a} q$ pour $(p, a, q) \in E$. L'ensemble des machines ayant pour alphabet d'entrée In et alphabet de sortie Out sera noté $\mathsf{Moore}(In, Out)$.

Dans son article fondateur, Moore [56] définit des machines *déterministes* : pour tout état $q \in Q$ et toute entrée $in \in In$, l'ensemble des arcs $\{(q, in, q') \in E\}$ est de cardinal égal à 1. La définition ci-dessus permet des machines *non-déterministes*. Un système en cours de développement est représenté par une machine de Moore non-déterministe qui lit des données de In et produit des données dans Out.

Exemple 2.1 : Si nous prenons In = **bool** et Out = **int**, un exemple de système (en l'occurrence déterministe) est celui convertissant la valeur **tt** en entrée en valeur 1 en sortie et convertissant **ff** en 0. La machine correspondante est $M_A = \langle \textbf{bool}, \textbf{int}, Q, \mathsf{out}, E, q_a \rangle$:

$$
\begin{aligned}
&Q = \{q_a, q_b\} \\
&\mathsf{out} = \{q_a \mapsto 0; q_b \mapsto 1\} \\
&E = \{(q_a, \textbf{ff}, q_a); (q_a, \textbf{tt}, q_b); (q_b, \textbf{tt}, q_b); (q_b, \textbf{ff}, q_a)\}.
\end{aligned}
$$

Cette machine admet la représentation graphique donnée en Fig. 2.1.

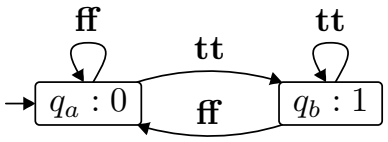


FIGURE 2.1 – Machine M_A

Étant donné un mot en entrée et un état de départ, il est possible d'effectuer une *séquence d'exécution*, produisant un ensemble de mots en sortie. L'espace d'état effectif de la machine est $Q \times Out$, où Q en est la composante *cachée* et Out la composante *observable*. La notion classique de langage reconnu peut être encodée en fixant $Out = \{REJECT, ACCEPT\}$ tel que $ACCEPT$ soit produit si et seulement si le mot entrant est reconnu.

Définition 2 (Séquence d'exécution, langage de sortie)**.** *Soit un mot d'entrée $w \in In^*$. Nous définissons les séquence d'exécution d'une machine de Moore $M = \langle In, Out, Q, \mathsf{out}, E, q_0 \rangle$ associée à un mot w comme les séquences appartenant à l'ensemble :*

$$\mathsf{Ex}_M(q_0, w) \subseteq Q \times (In \times Q)^*,$$

où ce dernier est défini par l'égalité suivante :

$$\mathsf{Ex}_M(q_0, w) = \{q_0.w[0].q_1.w[1].q_2 \ldots \mid \forall i, q_i \overset{w[i]}{\rightarrow} q_{i+1} \in E\}.$$

Le langage généré par la lecture de w depuis q_0 est l'ensemble des mots générés à partir de chaque séquence d'exécution par application de out *:*

$$\mathcal{L}^*_M(q_0, w) = \{\mathsf{out}(q_0).\mathsf{out}(q_1).\mathsf{out}(q_2) \ldots \mid q_0.a_0.q_1.a_1.q_2.a_2 \ldots \in \mathsf{Ex}_M(q_0, w)\}.$$

Exemple 2.2 : Nous donnons un exemple de séquence d'exécution pour la machine M_A donnée en Fig. 2.1. Soit w = **tt**.**ff**.**ff**.**tt** un mot d'entrée. Puisque M_A est déterministe, il n'existe qu'une unique séquence d'exécution pour w : q_a.**tt**.q_b.**ff**.q_a.**ff**.q_a.**tt**.q_b. Le langage de sortie associé est réduit à un unique mot :

$$
\begin{aligned}
\mathcal{L}^*_{M_A}(q_a, w) &= \{\mathsf{out}(q_a).\mathsf{out}(q_b).\mathsf{out}(q_a).\mathsf{out}(q_a).\mathsf{out}(q_b)\} \\
&= \{0.1.1.0.1\}.
\end{aligned}
$$

Noter que le 0 initial du mot de sortie est constant et indépendant de l'entrée.

2.2.3 Relations d'équivalence sur les états

Différentes notion d'équivalences peuvent être considérées pour les états d'une machine, avec différents degrés de précision. Dans les travaux qui suivent, le choix d'une relation 0d'équivalence s'avère crucial dans la définition du délai de séparabilité (cf. Sec. 2.3.4 p. 38). Une étude comparative de ces relations d'équivalences a été entreprise dans [68]. Selon cette étude, la relation d'équivalence identifiant le moins d'états dans notre cadre est la bisimilarité, que nous définissons ci-après.

2.2.3.1 Simulation et bisimulation

Nous commençons par définir la notion de simulation, qui est à la base de la notion de bisimulation et qui sera un outil fondamental de notre étude. Nous considérerons dans les définitions qui suivent une machine $M = \langle In, Out, Q, \mathsf{out}, E, Q_i \rangle$.

Informellement, un état q_2 est capable de simuler un autre état q_1 lorsque toute séquence d'exécution partant de q_1 peut être copiée depuis q_2. Nous donnons deux définitions équivalentes de la simulation, afin de mieux l'illustrer.

Définition 3 (Simulation)**.** *Soit $R \subseteq Q \times Q$ une relation sur les états de M. Nous définissons les conditions nécessaires et suffisantes pour que R soit une relation de simulation.*

Définition 3.1 *R est une relation de simulation si et seulement si, pour tout couple $(q_1, q_2) \in R$, la propriété suivante est vérifiée :*

$$\mathsf{out}(q_1) = \mathsf{out}(q_2) \ \wedge$$
$$\forall a \in In, \forall q_1' \in Q, (q_1 \xrightarrow{a} q_1' \in E \implies \exists q_2' \in Q, q_2 \xrightarrow{a} q_2' \in E \wedge (q_1', q_2') \in R.$$

Cette définition est la transposition de la description informelle donnée plus haut.

Définition 3.2 *Il est possible de donner une autre définition de la simulation, comme un morphisme entre systèmes de transitions. Observons que l' ensembles d'arcs $E \subseteq Q \times In \times Q$ peut être vu par associativité de $\times$ comme une relation $E \subseteq Q \times (In \times Q)$ ou $E \subseteq (Q \times In) \times Q$. Nous dirons alors que R est une relation de simulation si et seulement si :*

$$(R^{-1}; E) \subseteq (E; R^{-1}) \wedge \langle \mathsf{out}, \mathsf{out} \rangle (R) \subseteq Id.$$

Si il existe une relation de simulation R telle que $(q_1, q_2) \in R$, alors nous le noterons $q_1 \precsim_R q_2$, ou plus simplement $q_1 \precsim q_2$ selon nos besoins.

Remarquons que nos machines ayant potentiellement plusieurs états initiaux, il est direct de représenter des relations de simulation entre machines en calculant l'union de ces machines.

Exemple 2.3 : La figure 2.2 montre deux machines telles qu'il existe une relation de simulation R entre les états q_A et p_A. La relation en question est montrée par les flèches en pointillé :

$$R = \{(q_A, p_A); (q_B, p_B)\}.$$

La figure illustre le fait que la définition impose que pour tout chemin $p_A \ R^{-1} \ q_A \xrightarrow{in} q_0$, il doit exister un autre chemin $p_A \xrightarrow{in} p' \ R^{-1} \ q_1$ tel que $q_0 \ R \ q_1$, i.e. tel que le diagramme commute. Par exemple, le diagramme $p_A \ R^{-1} \ q_A \xrightarrow{\mathbf{tt}} q_b$ est fermé par $p_A \xrightarrow{\mathbf{tt}} p_B \ R^{-1} \ q_B$.

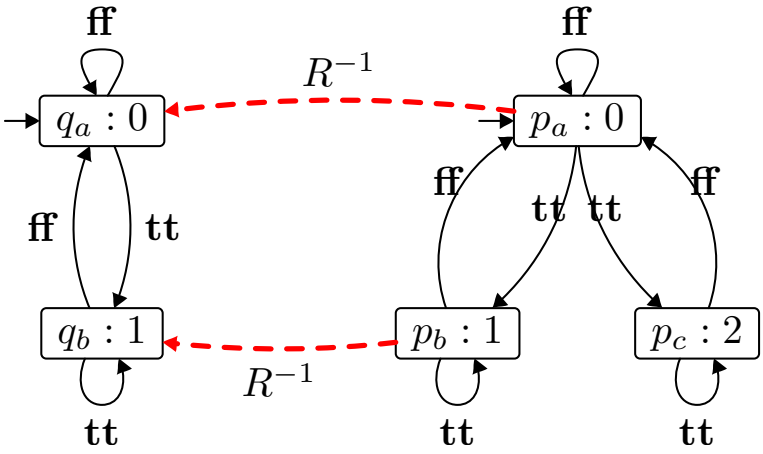


FIGURE 2.2 – Exemple de relation de simulation

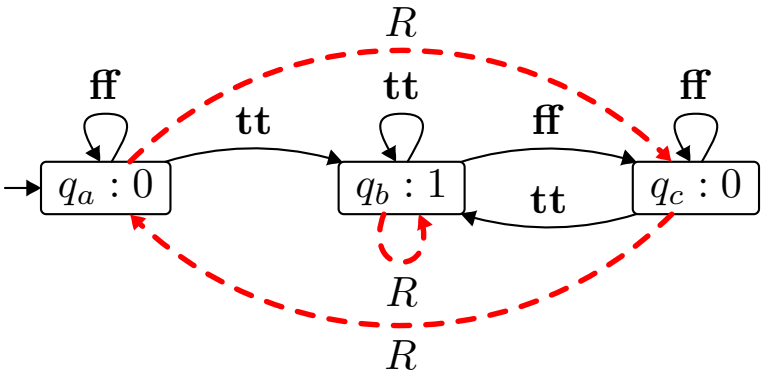


FIGURE 2.3 – Exemple de relation de bisimulation

La notion de bisimulation est définie à partir de la notion de simulation et la bisimilarité est définie comme l'union de toutes les bisimulations. Les définitions sont classiques, mais adaptées à notre cadre particulier.

Définition 4 (Bisimulation)**.** *Soit $R \subseteq Q \times Q$ une relation de simulation sur les états de M. Si R^{-1} est également une relation de simulation, alors R est une relation de bisimulation. L'union de toutes les bisimulations est notée $\sim$. $\sim$ est une relation d'équivalence. Nous pouvons définir explicitement le fait que R soit une bisimulation, comme dans le cas de la simulation :*

$$\begin{aligned}
&\forall (q_1, q_2) \in R, \forall a \in In, \mathsf{out}(q_1) = \mathsf{out}(q_2) \wedge \\
&\quad \forall q_1' \in Q, q_1 \xrightarrow{a} q_1' \in E \implies \exists q_2', q_2 \xrightarrow{a} q_2' \in E \wedge (q_1', q_2') \in R \wedge \\
&\quad \forall q_2' \in Q, q_2 \xrightarrow{a} q_2' \in E \implies \exists q_2', q_1 \xrightarrow{a} q_1' \in E \wedge (q_1', q_2') \in R.
\end{aligned}$$

Exemple 2.4 : La figure 2.3 montre un exemple de machine où les états q_a et q_c sont bisimilaires (cette machine sera montrée équivalente à la machine M_A en Fig. 2.1 p. 26). La relation de bisimulation R est illustrée sur la figure par les flèches en pointillé :

$$R = \{(q_a, q_c); (q_c, q_a); (q_b, q_b)\}.$$

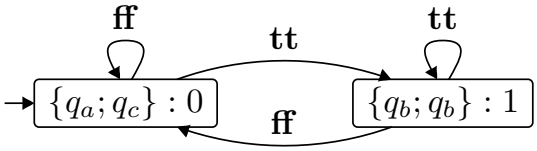


FIGURE 2.4 – Quotient par bisimulation

2.2.3.2 Extension aux séquences d'exécution

Ces relations d'équivalence s'étendent naturellement aux séquences d'exécution. Supposons que nous ayons la relation de simulation $q_1 \precsim q_2$. Alors il est possible de faire correspondre à toute séquence d'exécution $run_1 \in \mathsf{Ex}_{M_1}(q_1, w)$ une séquence d'exécution $run_2 \in \mathsf{Ex}_{M_2}(q_2, w)$, avec :

$$\begin{aligned} run_1 &= (q_1 = q_1^0).w[0].q_1^1.w[1].q_1^2.w[2]\ldots w[|w|-1].q_1^{|w|}; \\ run_2 &= (q_2 = q_2^0).w[0].q_2^1.w[1].q_2^2.w[2]\ldots w[|w|-1].q_2^{|w|} \end{aligned}$$

et telles que $\forall i, q_1^i \precsim q_2^i$. La preuve suit par application directe de la définition de simulation. Par extension, nous noterons cette situation $run_1 \precsim run_2$ dans le cas des séquences d'exécution prises individuellement et plus généralement $\mathsf{Ex}_{M_1}(q_1, w) \precsim \mathsf{Ex}_{M_2}(q_2, w)$. Le cas de la bisimulation est semblable.

2.2.4 Quotient par bisimulation et dépliage d'une machine

Il nous sera utile de travailler sur les machines en ignorant les différences entre les états jugés équivalents pour la bisimilarité. Dans la suite, nous supposons l'existence d'une machine $M = \langle In, Out, Q, \mathsf{out}, E, Q_i \rangle$.

2.2.4.1 Quotient par bisimulation

Le quotient par une relation de bisimulation R de M est la machine :

$$M/R = \langle In, Out, Q/R, \mathsf{out}', E', R(Q_i) \rangle$$

définie par :

- $Q/R = \{\mathcal{C}l \subseteq Q \mid \forall p, q \in \mathcal{C}l, p\ R\ q \ \wedge\ \forall p \notin \mathcal{C}l, q \in \mathcal{C}l, \neg(p\ R\ q)\}$,
- $\mathsf{out}'(\mathcal{C}l) = \mathsf{out}(q), q \in \mathcal{C}l$,
- $E' = \{(\mathcal{C}l_1, in, \mathcal{C}l_2) \mid (q_1, in, q_2) \in E \ \wedge\ q_1 \in \mathcal{C}l_1 \ \wedge\ q_2 \in \mathcal{C}l_2\}$, et
- $R(Q_i) = \{\{p \mid q_i\ R\ p\} \mid q_i \in Q_i\}$.

Par construction, M/R est bisimilaire à M.

Exemple 2.5 : La machine en Fig. 2.3 a pour quotient la machine en Fig. 2.4.

2.2.4.2 Dépliage d'une machine depuis un état

Le dépliage d'une machine depuis un état p est un arbre (de profondeur infinie) *structurellement égal* à tout dépliage d'un état q tel que $p \sim q$. Une définition de ces arbres (dans un contexte légèrement différent) peut être trouvée dans [68] p. 47, ou encore dans [3] où ils sont appelés *arbres de synchronisation*.

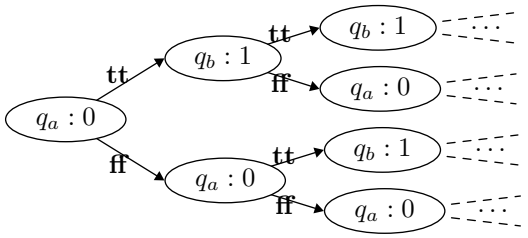


FIGURE 2.5 – Dépliage de M_A.

Définition 5 (Dépliage)**.** *Soient $In, Out \in \mathfrak{D}$ deux types de données. L'ensemble des dépliages sur In, Out est l'ensemble $Tree(In, Out)$, qui est la plus grande solution de l'équation :*

$$Tree(In, Out) = Out \times \wp_{fin}(In \times Tree(In, Out))$$

Le dépliage d'un état q d'une machine M est défini récursivement par la fonction $\mathcal{U}_M$:

$$\begin{array}{lcl} \mathcal{U}_M & : & Q \to Tree(In, Out) \\ \mathcal{U}_M(q) & = & \Big(\mathsf{out}(q),\ \Big\{(in, \mathcal{U}_M(q')) \mid \exists q \xrightarrow{in} q' \in E\Big\}\Big). \end{array}$$

(Noter que formellement, un dépliage est un ensemble non bien-fondé [5]).

Un exemple de dépliage de la machine M_A (donnée en Fig. 2.1 p. 26) est donné en Fig. 2.5. Afin de faciliter la lecture de ce dépliage, nous notons le nom des états de la machine d'origine dans chaque noeud.

Proposition 2.6 (Dépliages comme représentants canoniques des classes de bisimilarité)**.** *La proposition suivante est vérifiée :*

$$p \sim q \iff \mathcal{U}_M(p) \simeq \mathcal{U}_M(q)$$

Cette proposition importante affirme que pour tous états $p \sim q$, nous avons $\mathcal{U}_M(p) \simeq \mathcal{U}_M(q)$ et réciproquement. Les dépliages sont donc des représentants canoniques des classes d'équivalence d'états pour la bisimilarité. Une preuve (dans le cas similaire des machines de Mealy) peut être trouvée dans [3] p. 12 ; et dans le cas général dans [5]. De plus, il est direct de construire à partir de tout dépliage une machine à nombre infini d'états. Les définitions données pour les machines sont donc applicables aux dépliages.

2.3 Délai de séparabilité d'un système synchrone

Lors de la conception d'un système temps-réel embarqué, le respect des contraintes temporelles est primordial. Nous pouvons illustrer cet enjeu en considérant le comportement du modèle physique $\mathcal{S}_\Phi$ d'un système temps-réel, avec pour entrée une impulsion au temps t. Nous nous attendons naturellement à ce que l'effet de cette impulsion ne soit pas observable immédiatement en sortie : le temps de propagation du signal est non-nul. Ceci est illustré en Fig. 2.6. La réponse impulsionnelle de $\mathcal{S}_\Phi$ est (dans cet exemple) observable sur un *intervalle* $[t+\Delta_0; t+\Delta_1]$. En pratique, cet intervalle est également fonction de la finesse de nos instruments de mesure, mais en supposant que la précision soit suffisante, nous pouvons utiliser le temps de réponse

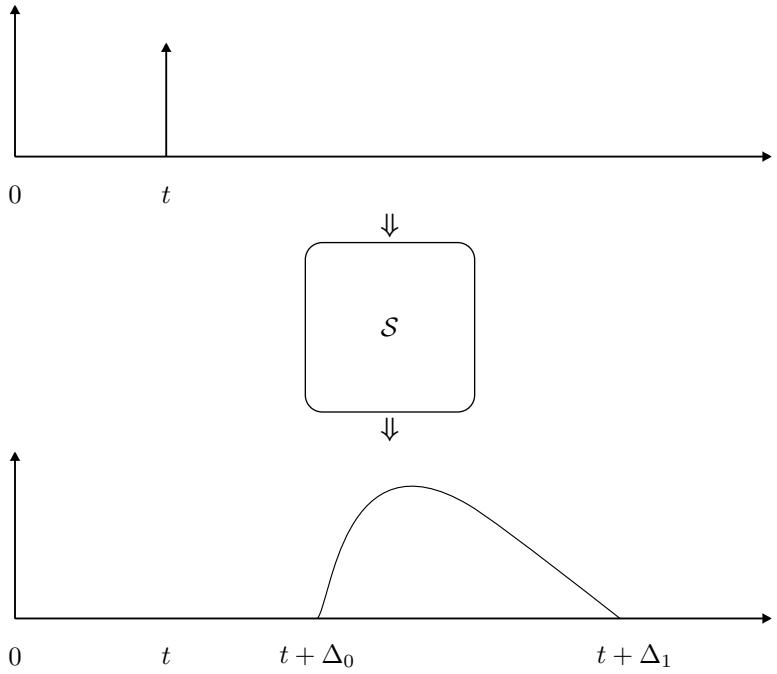


FIGURE 2.6 – Réponse impulsionnelle

"au plus tôt" $t + \Delta_0$ comme une mesure du "temps de réaction" de $\mathcal{S}_\Phi$. Si ce temps n'est pas compatible avec la dynamique du processus physique contrôle, le système pourra être considéré comme incorrect.

Nous cherchons à donner une caractérisation semblable dans le cas de systèmes discrets implémentés par des machines de Moore non-déterministes. Le premier problème est qu'il n'existe pas de caractérisation évidente de ce qu'est une "réponse" à une entrée : un programme informatique peut émettre une suite de données construite de façon arbitraire (en particulier, sans aucun rapport avec ses entrées). Cette idée est illustrée en Fig. 2.7, qui montre un système discret synchrone (par exemple composé de machines de Moore) calculant à partir d'une séquence d'entrées $m_0, m_1, m_2, \ldots$ une séquence de sorties $o_0, o_1, o_2, \ldots$. La figure suggère qu'une donnée en entrée particulière m_1 peut "influer" sur la valeur des sorties $o_3, o_5, o_6 \ldots$ d'indices arbitraires.

Le délai de séparabilité d'un système (dans un état particulier) peut alors être défini comme le nombre de ticks de l'horloge logique entre une entrée lue sur cet état et une réponse *significative*, c'est-à-dire fonctionnellement dépendante. Il est important de distinguer ce délai de séparabilité du concept de *temps de réaction* tel qu'il est compris dans le cadre de l'étude des systèmes synchrones. Pour le modèle synchrone fort, le temps de réaction est égal à 0 (cas des machines de Mealy) et à 1 pour le modèle synchrone faible (cas des machines de Moore). Notre notion de délai de séparabilité est inspiré du temps de réaction "physique". Ce délai se décompose en deux notions distinctes : le lien de *causalité* entre l'évènement en entrée et celui de sortie et le *délai* (en temps logique) entre ces deux évènements. Cette section se propose d'arriver par une démarche systématique à une définition formelle du délai de séparabilité. Nos travaux sont également liés aux recherches portant sur l'analyse du flot d'informations [27]. Le but est dans ces travaux de prouver que certaines données *privées* d'un programme (par exemple un mot de passe) ne sont pas déductibles du comportement *observable* dudit programme, même par des canaux d'information détournés (non-terminaison, temps d'exécution ...).

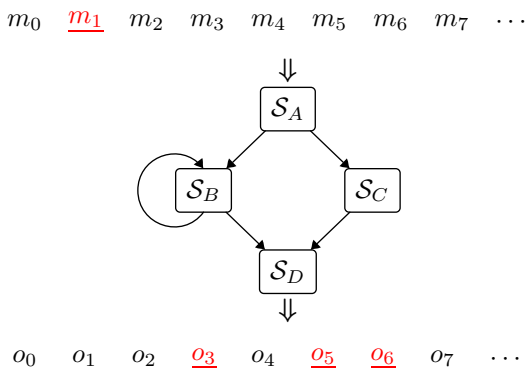


FIGURE 2.7 – Délai de séparabilité d'un système synchrone

Comme dit dans l'introduction, *notre* but est de montrer qu'il existe bel et bien un flot d'information entre les entrées et les sorties d'une machine, de borner le temps de transfert de cette information à travers cette machine et d'étudier le comportement de cette notion lors du développement par composition et par raffinement.

Deux outils formels ont déjà été utilisés pour étudier ce type de propriété : la *non-interférence* [37] et l'utilisation de la notion d'*entropie de Shannon* [65]. Nous nous inspirons principalement de la notion de *non-interférence*, qui est en fait une forme de dépendance fonctionnelle. Une étude basée sur l'entropie de Shannon sera discutée dans les perspectives de recherche, en conclusion.

2.3.1 Dépendance fonctionnelle et causalité

Nous illustrons notre démarche par un premier exemple. Noter que la méthode utilisée ici est très semblable à celle adoptée par Moore [56]. Soit un état de système q disposant d'une partie observable $Observable(q)$ et d'une partie cachée $Hidden(q)$. Nous supposerons pour la simplicité de l'exemple que le système est *déterministe*. Le système évolue sous l'action de deux fonctions A et B, agissant symboliquement comme des boutons poussoirs.

Soit également un observateur ignorant tout de $Hidden(q)$ et n'ayant comme moyens d'action que, d'une part, faire évoluer le système en utilisant A ou B et et d'autre part observer l'état du système au cours du temps.

Test de réactivité. Dans cette expérience, l'observateur a pour tâche de déterminer si presser A plutôt que B a une conséquence observable sur le futur du système. Une solution naïve est de vérifier si le nouvel état observable est différent du précédent.

Entrée	q	$\xrightarrow{A}$	q_A
État observable	o		o'

Si $o \neq o'$, l'observateur $\mathcal{O}$ considérera que le système semble bien avoir répondu à l'entrée A. Deux objections peuvent être faites à cette conclusion hâtive.

1. Le système dans l'état q avait peut-être prévu *à l'avance* de rendre observable o' ;

2. il n'y a pas de raison pour que l'effet observable survienne immédiatement après avoir pressé le bouton.

Afin d'arriver à une solution correcte, le principal point à prendre en compte est que l'état observable n'est pas seulement fonction du bouton pressé mais également de l'état interne. Pour répondre à la première objection, nous allons augmenter le pouvoir de l'observateur et lui permettre de procéder à la même expérience simultanément avec A et B (ceci correspond à l'*expérience multiple* de [56]).

Entrée	q	$\xrightarrow{A}$	q_A	q	$\xrightarrow{B}$	q_B
État observable	o		o'	o		o''

Si $o' \neq o''$, on en déduit que le système a bel et bien distingué entre le bouton A et le bouton B. Cette expérience est donc plus informative que la précédente.

Dans le cas contraire, savoir que $o' = o''$ ne nous permet de tirer aucune conclusion sur le comportement interne du système. La seconde objection nous dit que l'effet observable de cette entrée peut survenir après un nombre arbitraire de transitions.

Une réponse intuitive est d'itérer l'expérience sur q_A et q_B jusqu'à observer un effet observable (i.e. une différence entre les états observables), mais l'état observable devient alors corrélé aux autres choix A ou B effectués pendant l'expérience. Supposons que nous itérions l'expérience une fois de plus.

Entrée	q	$\xrightarrow{A}$	q_A	$\xrightarrow{x}$	q_A^1
État observable	o		o'		o_x

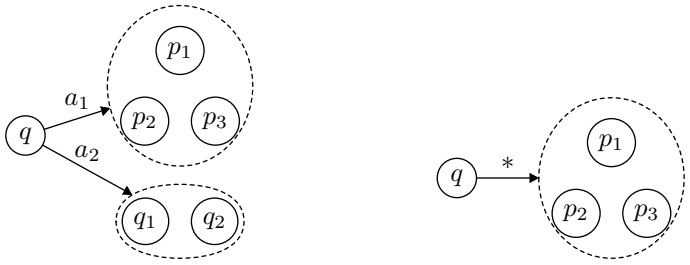
Entrée	q	$\xrightarrow{B}$	q_B	$\xrightarrow{y}$	q_B^1
État observable	o		o''		o_y

Ici, x et y sont à priori quelconques. Quelles que soient les valeurs de x et y, o_x est soit une constante, soit une conséquence observable de la première expérience où A a été pressé, soit de x (ces deux dernières éventualités ne sont pas mutuellement exclusives). De même, o_y est soit une constante, soit une conséquence de B, soit de y.

Si $x \neq y$ il est impossible d'en dire plus, mais si $x = y$ et $o_x \neq o_y$ alors nous *savons* que (o_x, o_y) est bien une conséquence observable du fait d'avoir initialement pressé A ou B. Nous pouvons généraliser ce résultat : si il existe une suite finie d'expériences (c'est-à-dire un mot sur $\{A, B\}$) permettant de distinguer les états q_A et q_B, alors q prend bien en compte son entrée initiale. En termes plus formels, q est "réactif" si q_A et q_B ne sont pas bisimilaires (ce qui dans le cas déterministe est équivalent à ce qu'ils n'aient pas les mêmes traces). Si nous replaçons cet exemple dans le cadre de l'exemple en Fig. 2.7, nous pouvons déterminer l'influence de l'entrée m_1 sur les sorties o_j en faisant varier le contenu de m_1 *uniquement*, et en observant quelles portions de la trace de sortie sont affectées par ces variations.

2.3.2 Réactivité d'un système synchrone

L'étude précédente montre la nécessité de faire *plus* qu'examiner la simple présence ou absence d'un "message" : il faut également en étudier le contenu. Afin de déterminer si un système répond à une entrée, il faut analyser la variabilité de l'ensemble de ses transitions *en fonction de*



(a) Fonction non-constante (b) Fonction constante

FIGURE 2.8 – État vu comme fonction

la valeur de l'entrée et montrer que la fonction ainsi calculée est non-constante. Cette section est dédiée à la définition d'une notion de réactivité d'un système synchrone modélisé par une machine de Moore non-déterministe. Dans cette section, nous travaillerons sur une machine $M = \langle In, Out, Q, \mathsf{out}, E, q_i \rangle$.

Cas des fonctions totales. Avant d'introduire la notion de réactivité, nous allons brièvement l'éclairer par son étude sur un cas simple, celui des fonctions totales. Nous procéderons ensuite à une définition adaptée aux machines de Moore.

Soient A et B deux ensembles et $f \in A \to B$ une fonction *totale* quelconque. Il est aisé de déterminer si f est constante : si en l'appliquant à tout son domaine A on obtient un ensemble image réduit à un point, la fonction f l'est. De façon duale, si il existe deux entrées *distinctes* $a_1 \neq a_2$ telles que $f(a_1) \neq f(a_2)$, alors f est non-constante. Nous appelons cette paire d'entrées (a_1, a_2) *paire séparante*.

Cas d'un état d'une machine de Moore. Nous allons procéder à une démarche similaire au cas des fonctions totales pour nos systèmes synchrones : un état $q \in Q$ sera réactif si et seulement si il est non-constant. La correspondance entre la notion d'*état* et celle de *fonction* mérite d'être explicitée. A tout état q correspond une fonction E_q associant toute entrée à un ensemble d'états directement atteignables :

$$\begin{array}{lcl} E_q & : & In \to \wp(Q) \\ E_q(a) & = & \{q' \mid \exists q \xrightarrow{a} q' \in E\}. \end{array}$$

La figure 2.8(a) montre un état q vu comme une fonction associant à l'entrée a_1 l'ensemble d'états $E_q(a_1) = \{p_1, p_2, p_3\}$ et à l'entrée a_2 l'ensemble d'états $E_q(a_2) = \{q_1, q_2\}$. Cette fonction est donc non-constante. La figure 2.8(b) montre le cas d'une fonction qui associe à toutes ses entrées le même ensemble d'état ; cette fonction est donc constante.

La *non-constance* de q peut alors s'écrire comme l'existence d'une telle paire d'entrées $(a_1, a_2) \in In \times In$ telles que les ensembles d'états $E_q(a_1)$ et $E_q(a_2)$ soient différents. Cette notion n'est cependant pas suffisante : deux états distincts $q_1 \neq q_2$ peuvent être jugés *équivalents* par une certaine relation (équivalence de traces, bisimilarité). Le choix d'une telle relation correspond à la capacité d'observation que l'on se donne sur les machines (cf. l'exemple du test de réactivité en Sec. 2.3.1 p. 32). Dans tous les cas, ce paramètre est pris en compte en quotientant les ensembles $E_q(a_1)$ et $E_q(a_2)$ par la relation d'équivalence choisie.

Dans nos travaux, nous donnons à notre observateur hypothétique la capacité de distinguer le comportement *branchant* des machines de Moore. En d'autres termes, nous utilisons la bisimilarité comme relation d'équivalence. Ceci est également justifié par le fait que la bisimilarité est

la plus précise relation d'équivalence sur les états qui n'influe pas sur ce que calcule la machine par quotient de l'espace d'état. Deux états seront donc considérés distincts si et seulement si ils sont non bisimilaires, ce que nous définissons ci-après.

Non-bisimilarité La définition naturelle de la non-bisimilarité est la simple négation logique de la relation de bisimilarité, notée $\not\sim$. Ceci revient à affirmer l'existence de *contre-exemples* à la bisimilarité. Afin d'obtenir des informations plus précises sur le nombre de transitions nécessaires pour prouver la non-bisimilarité, nous donnons une définition explicite de ces contre-exemples.

Proposition 2.7 (Preuve de non-bisimilarité)**.** *Soient $p, q \in Q$ deux états quelconques. Nous définissons $\mathsf{c{-}ex}(p,q)$ comme ensemble de tous les contre-exemples à la bisimilarité de p et q. Chaque contre-exemple est construit inductivement par les règles d'inférence suivantes :*

$$\textsc{base}\ \frac{\mathsf{out}(p) \neq \mathsf{out}(q)}{\mathsf{c{-}ex}(p,q)} \qquad \textsc{sym}\ \frac{\mathsf{c{-}ex}(q,p)}{\mathsf{c{-}ex}(p,q)}$$

$$\textsc{ind}\ \frac{\exists a \in In, \exists p' \in Q, \exists p \xrightarrow{a} p' \in E, \forall q' \in Q, q \xrightarrow{a} q' \in E \implies \mathsf{c{-}ex}(p',q')}{\mathsf{c{-}ex}(p,q)}$$

L'équivalence suivante est vérifiée :

$$p \not\sim q \iff \mathsf{c{-}ex}(p,q).$$

Démonstration. La preuve procède par calcul de la négation logique sur la définition de la bisimulation (Def. 4 p. 28). Il faut noter que la caractérisation inductive des contre-exemples ci-dessus n'est possible que parce que nous travaillons sur des machines à états finis et alphabets finis et donc avec un facteur de branchement fini pour les transitions. □

Exemple 2.8 (Non bisimilarité) **:** La figure 2.9 montre deux états p_0 et q_0 tels que $p_0 \not\sim q_0$. En l'occurrence, il existe au moins deux preuves distinctes de ce fait, comme indiqué par les chemins en gras. Les arbres de dérivation correspondants (simplifiés pour plus de lisibilité) sont les suivants :

$$\dfrac{p_0 \xrightarrow{\mathbf{ff}} p_1, q_0 \xrightarrow{\mathbf{ff}} q_1, \qquad \dfrac{p_1 \xrightarrow{\mathbf{tt}} p_5, q_1 \xrightarrow{\mathbf{tt}} q_5, \qquad \dfrac{3 \neq 8}{\mathsf{c{-}ex}(p_5,q_5)}\ \textsc{Base}}{\mathsf{c{-}ex}(p_1,q_1)}\ \textsc{Ind}}{\mathsf{c{-}ex}(p_0,q_0)}\ \textsc{Ind}$$

$$\dfrac{p_0 \xrightarrow{\mathbf{tt}} p_8, q_0 \xrightarrow{\mathbf{tt}} q_8, \qquad \dfrac{p_8 \xrightarrow{\mathbf{ff}} p_9, q_8 \xrightarrow{\mathbf{ff}} q_9, \qquad \dfrac{p_9 \xrightarrow{\mathbf{tt}} p_{11}, q_9 \xrightarrow{\mathbf{tt}} q_{11}, \qquad \dfrac{3 \neq 1}{\mathsf{c{-}ex}(p_{11},q_{11})}\ \textsc{Base}}{\mathsf{c{-}ex}(p_9,q_9)}\ \textsc{Ind}}{\mathsf{c{-}ex}(p_8,q_8)}\ \textsc{Ind}}{\mathsf{c{-}ex}(p_0,q_0)}\ \textsc{Ind}$$

Noter que les machines illustrées dans cet exemple sont déterministes. La bisimilarité correspond ici à l'équivalence de traces.

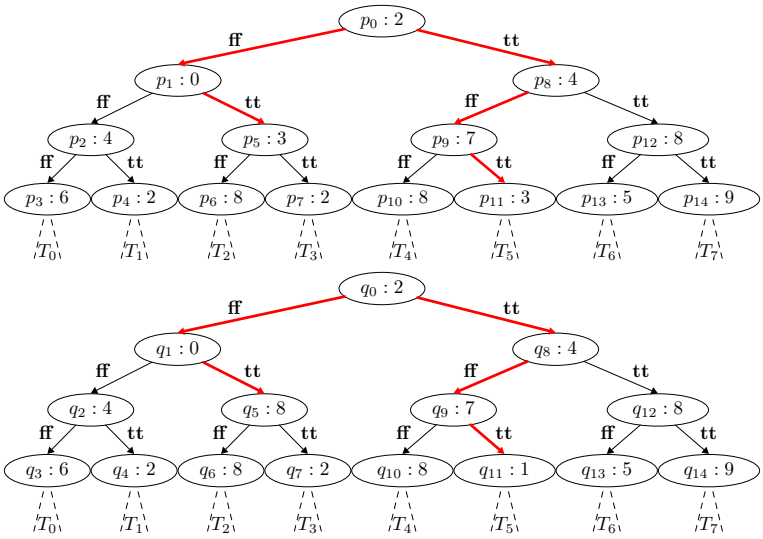


FIGURE 2.9 – Exemple d'états non bisimilaires.

La notion de non-bisimilarité va nous permettre de définir successivement les notions de réactivité, de paires séparantes, de séparateurs puis de délai de séparabilité.

Définition 6 (Réactivité d'un état, paires séparantes)**.** *Nous noterons le fait que* $q \in Q$ *est réactif par* $\mathsf{reactive}(q)$*. Le prédicat* reactive *est défini ainsi :*

$$\mathsf{reactive}(q) = \exists a_1, a_2 \in In, a_1 \neq a_2 \ \wedge \left(\begin{array}{l} \exists q \xrightarrow{a_1} q_1, \forall q \xrightarrow{a_2} q_2, q_1 \not\sim q_2 \vee \\ \exists q \xrightarrow{a_2} q_1, \forall q \xrightarrow{a_1} q_2, q_1 \not\sim q_2 \end{array} \right).$$

La paire d'entrées (a_1, a_2) *est dite* séparante*. L'ensemble des paires séparantes de* q *est noté* $\mathsf{SepPairs}(q)$*.*

2.3.3 Effets observables d'une entrée

La notion d'effet observable découle d'une étude approfondie de la réactivité. La propriété de réactivité découle de l'existence de contre-exemples à la bisimilarité. L'étude de ces contre-exemples permet de définir une notion d'effet observable, correspondant à une variation comportementale localisée dans le temps. Nous commençons par définir les séquences d'exécution séparantes.

Définition 7 (Séquences d'exécution séparantes)**.** *Soient* p, q *deux états tels que* $p \not\sim q$*. Nous avons donc* $\mathsf{c-ex}(p, q)$*. Soit* $h \in \mathsf{c-ex}(p, q)$ *une preuve* particulière *de la non-bisimilarité de* p *et* q*. Nous définissons* $\mathsf{SepEx}(h)$ *comme l'ensemble des séquences d'exécution séparantes de* h*,*

par induction sur h.

- $h = \dfrac{\mathsf{out}(p) \neq \mathsf{out}(q)}{\mathsf{c{-}ex}(p,q)}\ \text{BASE} \implies \mathsf{SepEx}(h) = \{(p,q)\}$

- $h = \dfrac{\mathsf{c{-}ex}(q,p)}{\mathsf{c{-}ex}(p,q)}\ \text{SYM}$

  $\implies \mathsf{SepEx}(h) = \{(seq_p,\ seq_q) \mid \exists h' \in \mathsf{c{-}ex}(q,p), (seq_q, seq_p) \in \mathsf{SepEx}(h')\}$

- $h = \dfrac{\exists a \in In, \exists p' \in Q, \exists p \xrightarrow{a} p' \in E, \forall q' \in Q, q \xrightarrow{a} q' \in E \implies \mathsf{c{-}ex}(p',q')}{\mathsf{c{-}ex}(p,q)}\ \text{IND}$

  $\implies \mathsf{SepEx}(h) = \bigcup_{p \xrightarrow{a} p'} \{(q.a.r_1,\ p.a.r_2) \mid \exists h' \in \mathsf{c{-}ex}(p',q'), \exists (r_1,r_2) \in \mathsf{SepEx}(h')\}$

Par extension, l'ensemble des séquences d'exécution séparantes de p *et* q *est l'union des séquences* $\mathsf{SepEx}(h)$ *pour toute preuve de non-bisimilarité* $h \in \mathsf{c{-}ex}(p,q)$ *:*

$$\mathsf{SepEx}(p,q) = \bigcup_{h \in \mathsf{c{-}ex}(p,q)} \mathsf{SepEx}(h)$$

Depuis tout arbre de dérivation prouvant $\mathsf{c{-}ex}(p,q)$, nous pouvons donc extraire une paire (au moins) de séquences d'exécution finies :

$$\begin{array}{cccccccc} & & \mathsf{Ind} & & \mathsf{Ind} & & & \mathsf{Base} \\ p & = & p_0 & \xrightarrow{a_0} & p_1 & \xrightarrow{a_1} \cdots \xrightarrow{a_{n-1}} & & p_n \\ & & \not\sim & & \not\sim & & & \not\sim \\ q & = & q_0 & \xrightarrow{a_0} & q_1 & \xrightarrow{a_1} \cdots \xrightarrow{a_{n-1}} & & q_n \end{array}$$

Chacune de ces paires de séquences d'exécutions correspond à un chemin entre la racine d'un contre-exemple et une feuille de ce même contre-exemple. Par construction, la paire d'états finale (p_n, q_n) est telle que $\mathsf{out}(p_n) \neq \mathsf{out}(q_n)$, ce que nous nommerons *effet observable*. De plus, chaque paire de séquences d'exécution séparantes est étiquetée par un mot $a_0.a_1 \ldots a_{n-1}$ que nous appellerons *mot séparateur*. Nous poursuivons en définissant formellement ces notions.

Définition 8 (Effets observables, séparateurs)**.**

1. *Si* p *et* q *sont tels que* $\mathsf{out}(p) \neq \mathsf{out}(q)$ *soit vérifié, alors* $(\mathsf{out}(p), \mathsf{out}(q))$ *est un effet observable.*
2. *Si* p *et* q *sont tels qu'il existe un couple* $(r_p, r_q) \in \mathsf{SepEx}(p,q)$ *de séquences d'exécution séparantes, alors les derniers symboles de sortie* $(\mathsf{out}(r_p[|r_p|-1]), \mathsf{out}(r_q[|r_q|-1]))$ *sont un effet observable de* p *et* q*.*
3. *Chaque séquence d'exécution séparante est étiquetée par un mot d'entrée appelé séparateur. L'ensemble des séparateurs de* p *et* q *est noté* $\mathsf{S}(p,q)$*. Nous pouvons faire des distinctions plus fines sur ces séparateurs. En particulier, le fait qu'un mot soit un séparateur ne garantit que l'*existence *de séquences d'exécution séparantes. L'ensemble des séparateurs pour lesquels toutes les séquences d'exécution induites sont séparantes est noté*

$\mathsf{S}_!(p,q)$ *(lire "séparateurs effectifs", en anglais "*must-separators*"). Ces ensembles sont définis par :*

$$\begin{array}{rcl} \mathsf{S}(p,q) & = & \{w \mid \exists r_1 \in \mathsf{Ex}(p,w), r_2 \in \mathsf{Ex}(q,w), (r_1,r_2) \in \mathsf{SepEx}(p,q)\} \\ \mathsf{S}_!(p,q) & = & \{w \mid \forall r_1 \in \mathsf{Ex}(p,w), \forall r_2 \in \mathsf{Ex}(q,w), (r_1,r_2) \in \mathsf{SepEx}(p,q)\} \end{array}$$

On a de façon triviale la relation :

$$\mathsf{S}_!(p,q) \subseteq \mathsf{S}(p,q).$$

4. *Un séparateur est minimal si et seulement si aucun de ses préfixes n'est un séparateur. Nous noterons ces séparateurs minimaux* $\mathsf{Minimal}(\mathsf{S}(p,q))$ *pour les séparateurs et* $\mathsf{Minimal}(\mathsf{S}_!(p,q))$ *pour les séparateurs effectifs.*

Noter que notre définition de c−ex permet d'avoir des différences de symboles de sortie pour chaque règle de dérivation et *pas seulement sur les feuilles*. Ceci nous permet de considérer de multiples effets observables générés au cours du temps par une même entrée.

Exemple 2.9 : La figure 2.10 montre un dépliage d'un état de nom $init$. Depuis cet état sont atteignables (par la paire séparante $(\mathbf{tt},\mathbf{ff}) \in \mathsf{SepPairs}(init)$) deux états q_0 et p_0. Le fait que ces deux états soient non-bisimilaires est prouvé par l'existence de trois séquences d'exécution séparantes :

$$\begin{array}{l|l} q_0 \xrightarrow{\mathbf{tt}} q_1 & p_0 \xrightarrow{\mathbf{tt}} p_1 \\ q_0 \xrightarrow{\mathbf{ff}} q_2 \xrightarrow{\mathbf{tt}} q_3 & p_0 \xrightarrow{\mathbf{ff}} p_2 \xrightarrow{\mathbf{tt}} p_3 \\ q_0 \xrightarrow{\mathbf{ff}} q_2 \xrightarrow{\mathbf{ff}} q_4 \xrightarrow{\mathbf{tt}} q_5 & p_0 \xrightarrow{\mathbf{ff}} p_2 \xrightarrow{\mathbf{ff}} p_4 \xrightarrow{\mathbf{tt}} p_5 \end{array}$$

Les séparateurs de q_0 et p_0 sont donc $\{\mathbf{tt}, \mathbf{ff}.\mathbf{tt}, \mathbf{ff}.\mathbf{ff}.\mathbf{tt}\}$, et appartiennent de surcroît à $\mathsf{S}_!(q_0,p_0)$. En revanche, les mots de l'ensemble $\mathbf{ff}^*$ ne sont pas des séparateurs.

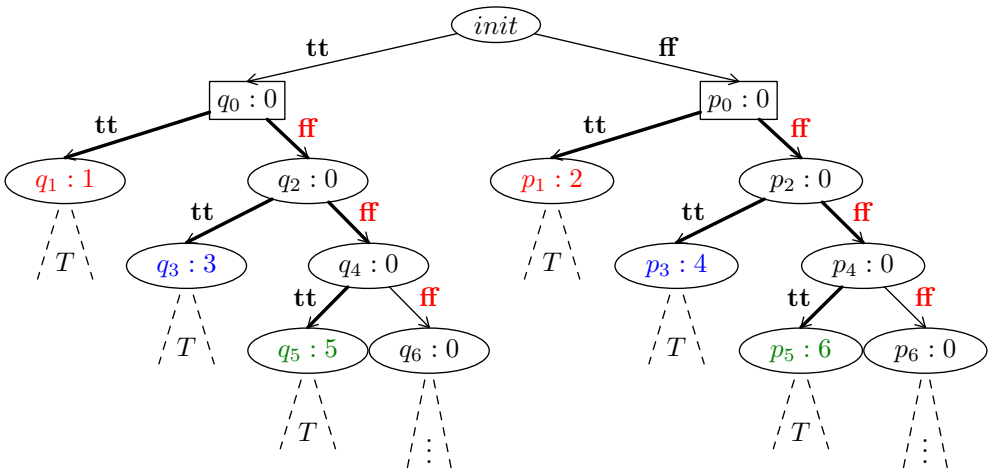


FIGURE 2.10 – Séparabilité

2.3.4 Délai de séparabilité

Le délai de séparabilité d'un état représente une estimation du temps nécessaire à une donnée en entrée pour avoir un effet observable en sortie d'une machine de Moore. Cette définition informelle s'accommode de multiples interprétations.

D'une manière générale, calculer le délai de séparabilité d'un *état* quelconque procède par le calcul du délai de séparabilité pour chaque paire séparante et par agrégation de ces délais de séparabilité intermédiaires à l'aide d'une opération à définir (minimum ou maximum, ...). De plus, calculer le délai de séparabilité pour une paire séparante particulière nécessite de calculer le délai de séparabilité entre les deux ensembles d'états atteignables par cette même paire, ce qui n'est possible qu'en définissant comment étendre le délai de séparabilité entre états à des ensembles.

Dans cette section, plutôt que de choisir une notion de délai de séparabilité de façon arbitraire, nous allons proposer un spectre de définitions cohérentes. Nous procéderons tout d'abord par l'étude du délai de séparabilité entre deux états, puis entre deux ensembles d'états. Enfin, nous étudierons comment calculer le délai de séparabilité d'un état à partir des délais de séparabilité pour chaque paire séparante.

2.3.4.1 Délai de séparabilité entre deux états

Puisque nous travaillons en temps logique, le temps auquel un effet observable se produit est son *indice* dans les traces de sortie associées. A partir du temps d'occurrence des effets observables, au moins deux notions de délai de séparabilité peuvent être construites.

- Nous pouvons adopter une approche optimiste et considérer la longueur de la plus courte séquence d'exécution jusqu'au premier effet observable.
- L'approche pessimiste est de considérer la profondeur maximale pour laquelle l'occurrence d'un effet observable est garantie.

La Fig. 2.11 montre des représentations abstraites des différences observables sur des dépliages de machines de Moore qui illustrent les notions exposées ci-dessus. La figure 2.11(a) présente le cas de machines de Moore pour lesquelles la première différence observable est atteignable après 10 transitions. La figure 2.11(b) le fait qu'il est possible que certaines séquences d'exécution n'entraînent l'occurrence d'aucun effet observable.

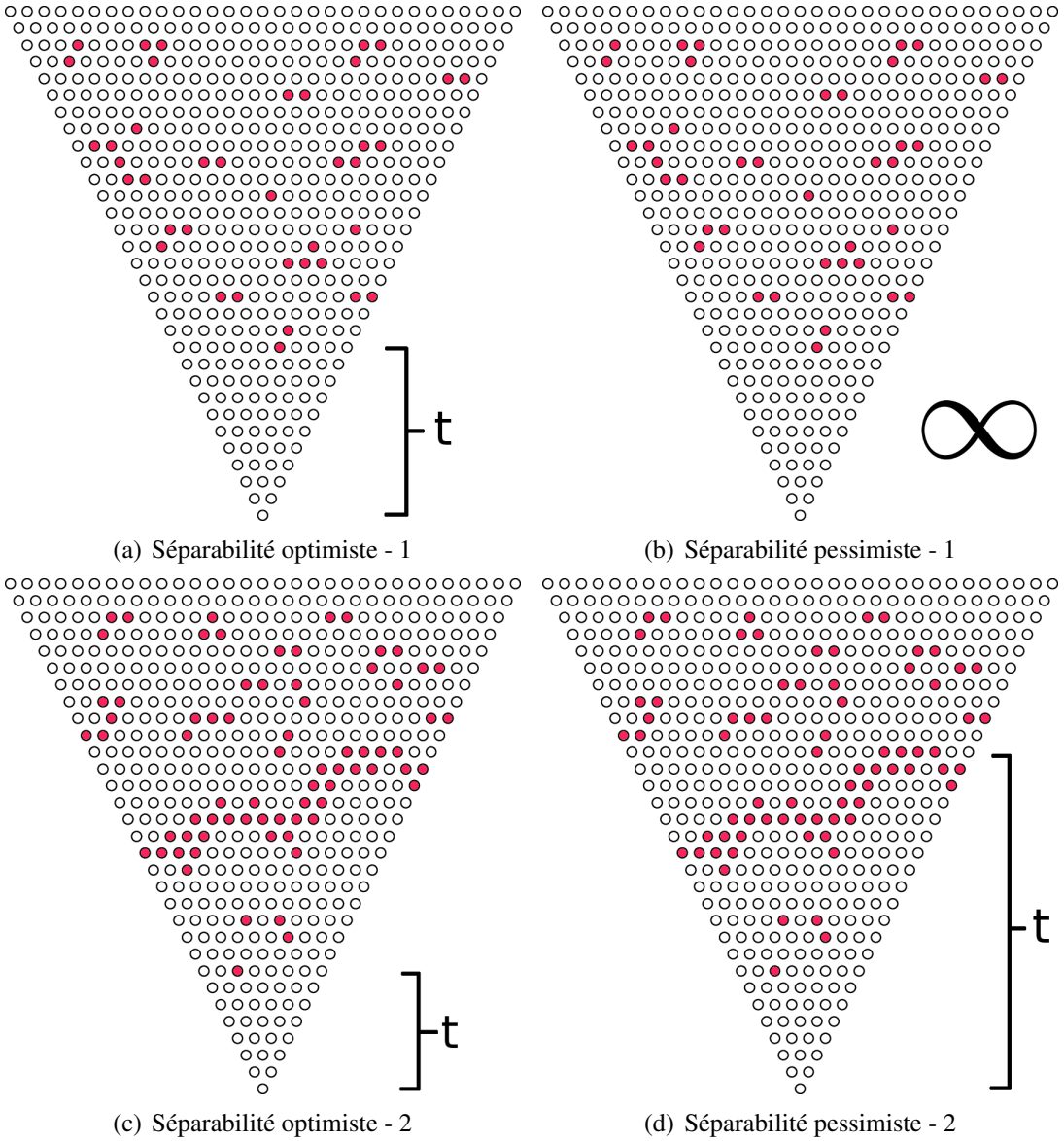
Définition 9 (Délai de séparabilité optimiste)**.** *Soient p, q deux états tels que $p \not\sim q$. Le délai de séparabilité optimiste de p et q, noté $\mathsf{septime}_{\mathsf{O}}(p, q)$, est défini comme la longueur de la plus petite séquence d'exécution induisant* potentiellement *un effet observable, c'est-à-dire la longueur du plus petit séparateur de p et q :*

$$\mathsf{septime}_{\mathsf{O}}(p, q) = \min \{|w| \mid w \in \mathsf{S}(p, q)\}.$$

Les figures 2.11(c) et 2.11(d) illustrent la seconde notion : toutes les séquences d'exécution permettent l'occurrence d'un effet observable et il est possible de considérer la séquence d'exécution sans effet observable la plus longue comme le délai de séparabilité de l'état initial. Exprimé tel quel, le délai de séparabilité pessimiste n'est pas directement exprimable en termes de séparateurs.

Définition 10 (Délai de séparabilité pessimiste)**.** *Soient p, q deux états tels que $p \not\sim q$. Le délai de séparabilité pessimiste de p et q est noté $\mathsf{septime}_{\mathsf{P}}(p, q)$ et est défini de façon duale au cas optimiste. Il s'agit de la longueur de la plus longue séquence d'exécution n'induisant* aucun *effet observable :*

$$\mathsf{septime}_{\mathsf{P}}(p, q) = \max \{|w| \mid \forall w' \in \mathsf{prefix}(w), w' \notin \mathsf{S}(p, q)\}$$



(a) Séparabilité optimiste - 1

(b) Séparabilité pessimiste - 1

(c) Séparabilité optimiste - 2

(d) Séparabilité pessimiste - 2

Chaque figure représente un dépliage, où chaque point représente un état atteignable depuis la racine. Les états mis en évidence symbolisent ceux dont la valeur de sortie est fonction de la valeur d'entrée lue à la racine. Les deux dépliages présentés sont étudiés dans les cas des délais optimistes et pessimistes. L'étude du cas pessimiste illustre le fait que tous les chemins doivent garantir un effet observable.

FIGURE 2.11 – Délai de séparabilité

2.3.4.2 Délai de séparabilité induit par une paire séparante

La section 2.3.2 montre que toute paire séparante induit un couple d'ensembles d'états atteignables. Il est possible d'étendre les notions de délai de séparabilité sur les états (définies précédemment) aux ensembles d'états par diverses méthodes.

Définition 11 (Délai de séparabilité induit par une paire séparante)**.**
Soit q un état et soit $(a_1, a_2) \in \mathsf{SepPairs}(q)$ une paire séparante. Nous rappelons que par définition, $E_q(a) = \{q' \mid \exists q \xrightarrow{a} q' \in E\}$.

– *Le délai de séparabilité optimiste correspondant à (a_1, a_2) est :*

$$\mathsf{septime_O}(a_1, a_2) = \mathsf{min}\ \{\mathsf{septime_O}(q_1, q_2) \mid q_1 \in E_q(a_1) \wedge q_2 \in E_q(a_2)\}\,.$$

– *Le délai de séparabilité pessimiste correspondant à (a_1, a_2) est :*

$$\mathsf{septime_P}(a_1, a_2) = \mathsf{max}\ \{\mathsf{septime_P}(q_1, q_2) \mid q_1 \in E_q(a_1) \wedge q_2 \in E_q(a_2)\}\,.$$

Le délai de séparabilité optimiste induit par une paire séparante correspond au minimum des délais de séparabilité pour toutes les paires d'états atteignables depuis un état q par cette paire séparante. De façon duale, le délai de séparabilité pessimiste est le maximum des délais de séparabilité pour toutes les paires d'états atteignables par cette paire séparante. Dans le cas pessimiste, il suffit donc qu'un seul couple d'états atteignable ait un délai infini pour que la paire séparante ait elle aussi un délai infini.

2.3.4.3 Délai de séparabilité d'un état

Le délai de séparabilité pour un état se déduit de manière similaire.

Définition 12 (Délai de séparabilité d'un état)**.**

1. *Le délai de séparabilité optimiste d'un état q est :*

$$\mathsf{septime_O}(q) = \mathsf{min}\ \{\mathsf{septime_O}(a_1, a_2) \mid (a_1, a_2) \in \mathsf{SepPairs}(q)\}.$$

2. *La variante pessimiste est :*

$$\mathsf{septime_P}(q) = \mathsf{max}\ \{\mathsf{septime_P}(a_1, a_2) \mid (a_1, a_2) \in \mathsf{SepPairs}(q)\}.$$

Nous pouvons définir les conditions nécessaires et suffisantes à ce que le délai *pessimiste* ne soit pas infini. La définition ci-dessus nous permet d'affirmer que le délai pessimiste est fini si et seulement si toutes les paires séparantes induisent un délai de séparabilité fini. La définition 11 p. 41 nous indique qu'une paire séparante (a_1, a_2) induit un délai fini si et seulement si toutes les paires d'états $(q_1, q_2) \in E_q(a_1) \times E_q(a_2)$ atteignables par elle induisent un délai fini. Enfin, le délai d'une paire d'états (q_1, q_2) est fini si et seulement si tous les mots d'entrée infinis sont préfixés par un séparateur de (q_1, q_2). Le délai de séparabilité pessimiste induit donc des contraintes considérables sur les états considérés.

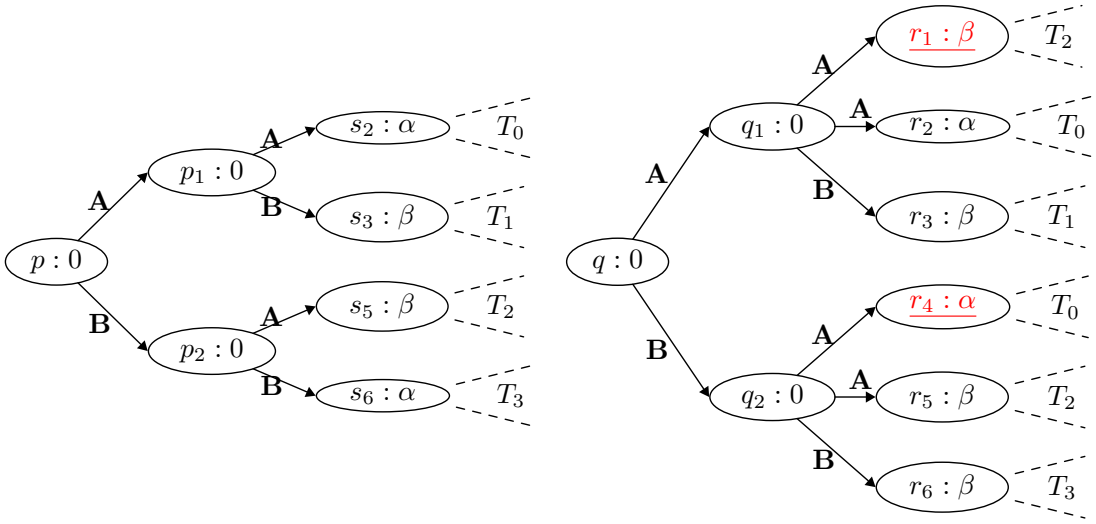


FIGURE 2.12 – Contre-exemple à la conservation des séparateurs non-effectifs par simulation

2.4 Effets observables par simulation et simulation inverse

Dans cette section, nous supposons l'existence d'une machine $M = \langle In, Out, Q, \mathsf{out}, E, q_0 \rangle$. Considérons des états $q_1, q_2, p_1, p_2 \in Q$ tels que $p_1 \precsim q_1$ et $p_2 \precsim q_2$. Soit $\mathcal{P} \subseteq Q \times Q$ une propriété quelconque, nous pouvons étudier sa conservation dans deux situations duales :

- par simulation : si $\mathcal{P}(p_1, p_2)$, est-ce que $\mathcal{P}(q_1, q_2)$?
- Par simulation inverse : si $\mathcal{P}(q_1, q_2)$, est-ce que $\mathcal{P}(p_1, p_2)$?

Plus précisément, nous allons étudier la conservation de certains ensembles de mots générant des effets observables, à commencer par les séparateurs.

2.4.1 Simulation

Le cas de la simulation est direct : les séparateurs ne sont en général pas conservés.

Proposition 2.10 (Non-conservation des séparateurs par simulation)**.** *Si $w \in \mathsf{S}(p_1, p_2)$, alors il est possible de construire q_1, q_2 tels que $p_1 \precsim q_1$, $p_2 \precsim q_2$ et $w \notin \mathsf{S}(q_1, q_2)$.*

Démonstration. Il suffit de compléter les dépliages de p_1 et p_2 de façon à ce qu'ils deviennent égaux. La figure 2.12 montre un exemple d'une telle opération. La preuve que dans chaque cas, $p_i \precsim q_i$ est directe. □

Un corollaire de cette proposition est que la propriété de réactivité et le délai de séparabilité ne sont pas conservés non plus.

2.4.2 Simulation inverse

Nous nous intéressons maintenant au cas de la simulation inverse. Nous étudions les séparateurs puis les paires séparantes.

2.4.2.1 Conservation des séparateurs effectifs

Il s'avère que le plus grand ensemble de séparateurs conservé par simulation inverse correspond aux séparateurs effectifs.

Proposition 2.11 (Conservation et optimalité des séparateurs effectifs)**.** *Soit $S_{\precsim}(q_1, q_2)$ l'ensemble des séparateurs préservés par simulation :*

$$S_{\precsim}(q_1, q_2) = \{w \in \mathsf{S}(q_1, q_2) \mid \forall p_1 \precsim q_1, \forall p_2 \precsim q_2, w \in \mathsf{S}(p_1, p_2)\}.$$

La proposition suivante est vérifiée :

$$S_{\precsim}(q_1, q_2) = \mathsf{S}_!(q_1, q_2).$$

Démonstration.

1. La preuve de $\mathsf{S}_!(q_1, q_2) \subseteq S_{\precsim}(q_1, q_2)$ suit du fait que toutes les séquences d'exécution d'un séparateur effectif sont *séparantes*. Une propriété de la simulation inverse est de restreindre l'ensemble des séquences d'exécution. Les séparateurs effectifs sont donc conservés par simulation.
2. Nous prouvons maintenant $S_{\precsim}(q_1, q_2) \subseteq \mathsf{S}_!(q_1, q_2)$. Soit $w \in S_{\precsim}(q_1, q_2)$, i.e. $w \in \mathsf{S}(q_1, q_2)$ et $\forall p_1 \precsim q_1, p_2 \precsim q_2, w \in \mathsf{S}(p_1, p_2)$. Nous raisonnons par l'absurde : supposons $w \notin \mathsf{S}_!(q_1, q_2)$. Ceci implique l'existence d'une paire de séquences d'exécution *non-séparantes* :

   $$(run_1, run_2) \in \mathsf{Ex}(q_1, w) \times \mathsf{Ex}(q_2, w).$$

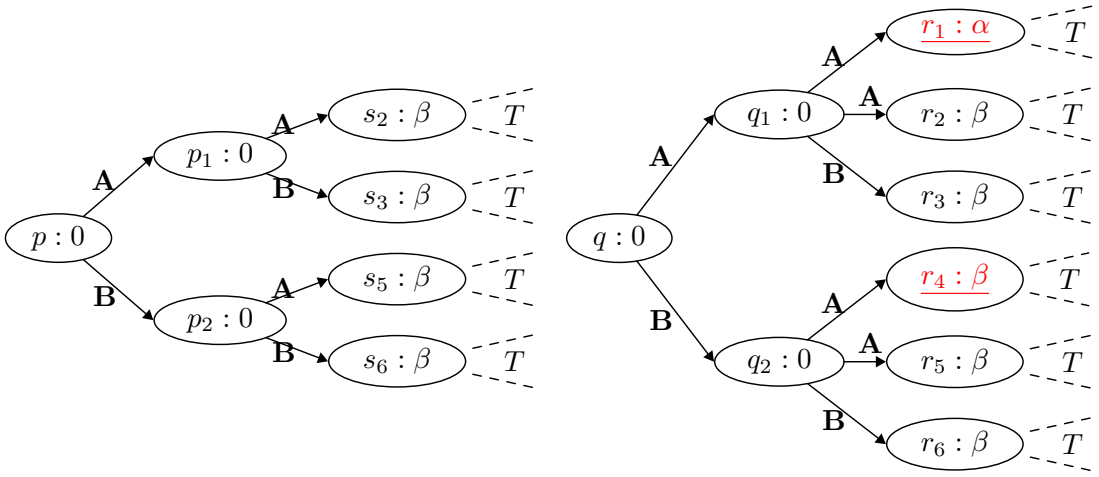
   A l'aide de cette hypothèse, nous allons montrer que w n'est pas préservé par simulation, par analyse par cas sur w. De cette contradiction découlera la proposition initiale.
   - (Cas de base) Le cas $w = \epsilon$ est contradictoire avec $w \notin \mathsf{S}_!(q_1, q_2) \wedge w \in \mathsf{S}(q_1, q_2)$.
   - (Cas inductif) Nous avons $w = a.w'$, avec $\exists q_1 \xrightarrow{a} q_1', \exists q_2 \xrightarrow{a} q_2', q_1' \sim q_2'$. Nous pouvons construire des états $p_1 \precsim q_1$ et $p_2 \precsim q_2$ tels que $p_1 \xrightarrow{a} q_1'$ et $p_2 \xrightarrow{a} q_2'$ soient les uniques transitions par a. Puisque $q_1' \sim q_2'$, $w \notin \mathsf{S}(p_1, p_2)$ et la propriété initiale est prouvée.

□

Exemple 2.12 : La Fig. 2.13 montre un exemple de non-conservation de séparateurs non-effectifs. Cette figure montre deux dépliages montrant des états candidats tels que $p_1 \precsim q_1$ et $p_2 \precsim q_2$. Dans la machine simulante, les états q_1 et q_2 ont pour séparateur potentiel le mot A, avec comme séquences d'exécution séparantes $q_1.A.r_1$ et $q_2.A.r_4$. La simulation permet d'effacer ces transitions et le dépliage de la machine simulée est tel que p_1 est équivalent à p_2.

2.4.2.2 Mots induisant un effet observable et conservés par simulation inverse

Nous nous intéressons maintenant au cas de mots d'entrée arbitraires. En particulier, nous voulons caractériser le plus grand ensemble de mots d'entrée garantissant l'occurrence d'un effet observable (à un temps potentiellement variable) et qui conserve cette propriété par simulation inverse. Il s'avère que ces mots sont précisément ceux qui induisent des langages de sortie disjoints. Dans cette section, nous donnons une définition de cet ensemble en termes de séparateurs et prouvons la propriété de conservation par simulation.



Machine simulée **Machine simulante**

FIGURE 2.13 – Contre-exemple à la conservation des séparateurs non-effectifs par simulation inverse

Etant donnés deux états $q_1, q_2 \in Q$, les mots d'entrée induisant des langages de sortie disjoints sont définis par :

$$Disj(q_1, q_2) = \{w \mid \mathcal{L}^*(q_1, w) \bigcap \mathcal{L}^*(q_2, w) = \varnothing\}.$$

Il est aisé de voir que pour tout $w \in Disj(q_1, q_2)$ et tout suffixe quelconque $suffix \in In^*$, nous avons $w.suffix \in Disj(q_1, q_2)$. De façon duale, pour tout $w \in Disj(q_1, q_2)$ il existe un préfixe de taille minimale $prefix \in Disj(q_1, q_2)$ tel que tout préfixe strictement plus petit n'induit pas de langages disjoints. Nous appelons ces mots irréductibles des "pseudo-séparateurs".

Définition 13 (Pseudo-séparateurs, définition directe)**.** *L'ensemble des pseudo-séparateurs, noté* $\mathsf{S}_*(q_1, q_2)$*, est défini par :*

$$\mathsf{S}_*(q_1, q_2) \;=\; \{w \in Disj(q_1, q_2) \mid \forall w_0 \in In^*, \forall w_1 \in In^+, w = w_0.w_1 \implies w_0 \notin Disj(q_1, q_2)\}.$$

Par définition, nous avons $Disj(q_1, q_2) = \{w_A.w_B \mid w_A \in \mathsf{S}_*(q_1, q_2), w_b \in In^*\}$. Nous allons donner une définition des pseudo-séparateurs ne reposant pas sur l'ensemble $Disj(q_1, q_2)$ et qui nous permettra de montrer leur conservation par simulation. Nous commençons par définir inductivement l'ensemble $\overline{\mathsf{S}_*}(q_1, q_2)$:

$$\frac{\text{PSEUDO-MUST}\quad w \in \mathsf{S}_!(q_1, q_2)}{w \in \overline{\mathsf{S}_*}(q_1, q_2)} \qquad \frac{\text{PSEUDO-IND}\quad w \in \mathsf{S}(q_1, q_2) \;\wedge\; \forall p_1 \precsim q_1, \forall p_2 \precsim q_2, \exists w_A, w_B, w = w_A.w_B \wedge w_A \in \overline{\mathsf{S}_*}(p_1, p_2)}{w \in \overline{\mathsf{S}_*}(q_1, q_2)}$$

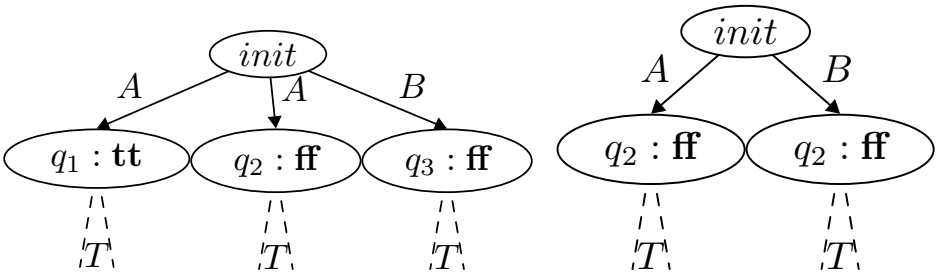
L'ensemble $\overline{\mathsf{S}_*}(q_1, q_2)$ correspond aux séparateurs qui par simulation inverse sont toujours préfixés par un séparateur. Les mots de cet ensemble garantissent donc bien la possibilité d'un effet observable. Si l'espace d'états ainsi que les alphabets sont finis alors nous pouvons définir cet ensemble *inductivement*, puisque la relation de simulation inverse est bien fondée. En d'autres termes, toute simulation inverse est une composition finie de simulations inverses "atomiques", car étant donnée une machine à états finis, il n'est possible de supprimer qu'un nombre fini d'arcs. Le fait que les pseudo-séparateurs sont conservés par simulation inverse et garantissent l'occurrence d'un effet observable est exprimé par la proposition suivante.

Proposition 2.13 (Caractérisation des pseudo-séparateurs en terme de séparateurs)**.**
L'inclusion suivante est vérifiée : $\mathsf{S}_*(q_1, q_2) \subseteq \overline{\mathsf{S}_*}(q_1, q_2)$.

La preuve consiste à montrer que tout pseudo-séparateur est également un séparateur. Cette preuve repose de façon cruciale sur la minimalité des pseudo-séparateurs, sans quoi nous ne pourrions pas prouver que les feuilles des contres-exemples étiquettés par ce pseudo-séparateur sont bien des effets observables (considérer l'exemple d'un mot d'entrée induisant des mots de sortie disjoints dès son premier symbole, les symboles restants ne produisant pas d'effets observables).

Démonstration. Soit $w \in \mathsf{S}_*(q_1, q_2)$. Nous procédons par analyse par cas sur la nature de w.

- Si $w \in \mathsf{S}_!(q_1, q_2)$, la preuve est terminée (cas PSEUDO-MUST).
- Dans le cas contraire, nous commençons par montrer $w \in \mathsf{S}_*(q_1, q_2) \implies w \in \mathsf{S}(q_1, q_2)$ par induction sur la séquence $w = u_0, w = w[0].u_1, w = w[0].w[1].u_2 \ldots$ des suffixes u_i de taille décroissante de w. Nous maintenons au cours de l'induction un ensemble R_i de couples de séquences d'exécution générées par $w[0] \ldots w[i-1]$ depuis respectivement q_1 et q_2 et qui génèrent des mots identiques. Initialement, $R_0 = \{(q_1, q_2)\}$ (séquences d'exécution de longueur 0).
  - (Cas de base.) Nous avons $u = w[|w| - 2]$. Soit un couple quelconque de séquences d'exécutions de $R_{|w|}$:
  $$q_1 \xrightarrow{w[0]} \ldots \xrightarrow{w[|w|-2]} p_1,$$
  $$q_2 \xrightarrow{w[0]} \ldots \xrightarrow{w[|w|-2]} p_2.$$
  Par invariant de $R_{|w|}$, ces séquences induisent les mêmes mots de sortie. Afin de satisfaire l'hypothèse $w \in \mathsf{S}_*$, la propriété suivante est nécessairement vérifiée :
  $$u \in \mathsf{S}_!(p_1, p_2).$$
  Dans le cas contraire, les langages de sortie induits par w ne seraient pas disjoints. Nous pouvons donc affirmer $p_1 \not\sim p_2$ pour tous les états finaux des séquences d'exécution de $R_{|w|}$.
  - (Cas inductif.) Soit u_i le suffixe courant. Soit un couple quelconque de séquences d'exécution de R_i :
  $$q_1 \xrightarrow{w[0]} \ldots \xrightarrow{w[i-1]} p_1,$$
  $$q_2 \xrightarrow{w[0]} \ldots \xrightarrow{w[i-1]} p_2.$$
  Considérons une transition quelconque $p_1 \xrightarrow{u_i[0]} p_1'$. Prouvons l'assertion suivante :
  $$\textsc{Ind}\ \frac{\exists p_1', \exists p_1 \xrightarrow{u_i[0]} p_1', \forall p_2', p_2 \xrightarrow{u_i[0]} p_2' \implies p_1' \not\sim p_2'}{p_1 \not\sim q_2}$$
  Pour chaque couple (p_1', p_2'), deux cas peuvent se présenter :
  - Si $\mathsf{out}(p_1') \neq \mathsf{out}(p_2')$, la preuve est conclue par application de la règle BASE.
  - Sinon, nous avons $\mathsf{out}(p_1') = \mathsf{out}(p_2')$. Il doit donc nécessairement exister un préfixe de u_i tel que la lecture de ce préfixe depuis (p_1', p_2') génère des langages disjoints. La preuve suit donc par application de l'hypothèse d'induction, avec les séquences d'exécution suivantes dans R_{i+1} :
  $$\left(q_1 \xrightarrow{w[0]} \ldots p_1',\ q_2 \xrightarrow{w[0]} \ldots p_2'\right) \in R_{i+1}.$$



(a) Paire séparante A, B (b) Absence de paire séparante

FIGURE 2.14 – Perte d'une paire d'entrée séparante

Nous avons donc $w \in \mathsf{S}_*(q_1, q_2) \implies w \in \mathsf{S}(q_1, q_2)$. Soient $p_1 \precsim q_1$ et $p_2 \precsim q_2$ deux états. Par simulation inverse, les langages restent disjoints. De ce fait, il existe un préfixe u de w tel que $u \in \mathsf{S}_*(p_1, p_2)$ permettant de clore la preuve. Le cas symétrique est direct et est omis.

□

Puisque la bisimilarité implique l'équivalence de traces, la propriété que les langages de sortie des états considérés soient différents implique la non-bisimilarité. Les pseudo-séparateurs induisent une propriété encore plus forte : les langages de sortie produits par un pseudo-séparateur sont *disjoints*. Ceci provient du fait qu'un pseudo-séparateur *garantit* l'occurrence d'un effet observable, quels que soient les choix non-déterministes effectués par la machine considérée.

2.4.2.3 Conséquences sur la préservation des paires séparantes

La proposition 2.11 p. 43 montre que les séparateurs ne sont en général pas conservés par simulation inverse. Ce résultat s'étend directement aux paires séparantes, comme montré dans la figure 2.14. La proposition 2.13 sur la conservation des pseudo-séparateurs nous permet de conclure que pour tout état q, une paire séparante (a_1, a_2) de q est conservée par toute simulation inverse si et seulement si les conditions énoncées dans la définition qui suit sont vérifiées.

Définition 14 (Paires séparantes stables)**.** *L'ensemble de paires séparantes conservées par simulation inverses sera nommé* paires séparantes stables *et sera noté* $\mathsf{SepPairs}_*(q)$ *:*

$$(a_1, a_2) \in \mathsf{SepPairs}_*(q) \iff \forall q_1, \forall q_2, q \xrightarrow{a_1} q_1 \wedge q \xrightarrow{a_2} q_2 \implies \exists w \in \mathsf{S}_*(q_1, q_2).$$

2.4.2.4 Conséquences sur la préservation du délai de séparabilité

Les résultats précédents nous donnent des conditions nécessaires et suffisantes à la conservation des séparateurs et des paires séparantes par simulation inverse. Nous appliquons maintenant ces derniers résultats à l'étude de la préservation des différentes notions de délai de séparabilité par simulation inverse.

Délai de séparabilité optimiste. Soient les états $q_1, q_2 \in Q$ tels que $q_1 \not\sim q_2$. Le délai de séparabilité optimiste de q_1 et q_2 est par définition :

$$\mathsf{septime}_{\mathsf{O}}(q_1, q_2) = \min \{|w| \mid w \in \mathsf{S}(q_1, q_2)\}.$$

Nous avons vu que seuls les pseudo-séparateurs (dont les séparateurs effectifs) garantissent la possibilité d'un effet observable après simulation inverse. Nous pouvons donc affirmer que le délai de séparabilité optimiste d'un couple d'états (q_1, q_2) ne peut croître par simulation inverse au delà de la longueur de son plus court pseudo-séparateur :

$$\forall p_1, p_2 \in Q, p_1 \precsim q_1 \wedge p_2 \precsim q_2 \implies \mathsf{septime}_{\mathsf{O}}(p_1, p_2) \leq \mathsf{min}\,\{|w| \mid w \in \mathsf{S}_*(q_1, q_2)\}.$$

La proposition 2.13 montre en effet que tout pseudo-séparateur donne lieu par simulation inverse à un préfixe pseudo-séparateur.

Cependant, ces propriétés ne se transposent pas directement au délai de séparabilité. En effet, il est direct de montrer que l'ensemble des états atteignables depuis un état quelconque se réduit par simulation inverse, ce qui implique que des paires d'états (q_1, q_2) dotées de pseudo-séparateurs peuvent tout simplement disparaître. Les conclusions suivantes peuvent être tirées de ces observations et des résultats précédents.

- En règle générale, le délai de séparabilité optimiste d'une paire séparante (a_1, a_2) n'est pas conservé par simulation inverse.
- Si nous considérons une paire séparante *stable* $(a_1, a_2) \in \mathsf{SepPairs}_*(q)$, alors nous pouvons borner $\mathsf{septime}_{\mathsf{O}}(a_1, a_2)$ par le haut.

Nous rappelons que le délai de séparabilité optimiste d'un état q, noté $\mathsf{septime}_{\mathsf{O}}(q)$ est défini comme suit :

$$\mathsf{septime}_{\mathsf{O}}(q) = \mathsf{min}\,\{\mathsf{septime}_{\mathsf{O}}(a_1, a_2) \mid (a_1, a_2) \in \mathsf{SepPairs}(q)\}.$$

Les résultats portant sur les paires séparantes de q se généralisent par conséquent directement au délai de séparabilité optimiste d'un état q.

- En général, $\mathsf{septime}_{\mathsf{O}}(q)$ n'est pas conservé et peut devenir infini.
- Il existe une paire séparante stable $(a_1, a_2) \in \mathsf{SepPairs}_*(q)$ si et seulement si $\mathsf{septime}_{\mathsf{O}}(q)$ est fini. Nous pouvons alors le majorer :

$$\forall p \precsim q, \mathsf{septime}_{\mathsf{O}}(p) \leq \mathsf{max} \left\{ |w| \;\middle|\; \begin{array}{l} \exists (a_1, a_2) \in \mathsf{SepPairs}_*(q), \\ \exists q_1, q_2 \in Q, \exists q \xrightarrow{a_1} q_1, q \xrightarrow{a_2} q_2 \in E, \\ w \in \mathsf{Minimal}(\mathsf{S}_*(q_1, q_2)) \end{array} \right\}.$$

  Noter que le fait de prendre les pseudo-séparateurs de tailles minimales n'est utile que pour obtenir un majorant plus fin.

Délai de séparabilité pessimiste. Soient les états $q_1, q_2 \in Q$ tels que $q_1 \not\sim q_2$. Le délai de séparabilité pessimiste existe si et seulement si tous les mots d'entrée infinis induisent un effet observable en temps fini (Def. 10 p. 39). Puisque les séparateurs ne sont pas conservés par simulation inverse, le délai de séparabilité pessimiste n'est comme sa variante optimiste pas conservé par simulation inverse. Nous pouvons cependant identifier les circonstances sous lesquelles il devient possible de montrer certains résultats de conservation.

Les pseudo-séparateurs garantissent l'occurrence d'un effet observable en un temps borné par leur longueur (Sec. 2.4.2.2 p. 43). Nous pouvons tenter de les utiliser pour implémenter une restriction du délai de séparabilité pessimiste conservée par simulation inverse. Nous dirons que deux états p et q sont *fortement séparables*, noté $p \rangle\!\langle q$, si et seulement si tous les mots d'entrée sont préfixés par un pseudo-séparateur. Posons la définition correspondante :

$$p \rangle\!\langle q \iff (\forall w \in In^\omega, \exists w' \in prefix(w), w' \in \mathsf{S}_*(p, q))\,.$$

Nous pouvons calculer $\mathsf{septime_P}(q_1, q_2)$ si $p \between q$:

$$\mathsf{septime_P}^*(q_1, q_2) = \mathsf{max}\ \{|w| \mid w \in \mathsf{S}_*(q_1, q_2)\}.$$

En utilisant la caractérisation inductive des pseudo-séparateurs (p. 44), nous voyons immédiatement que le délai de séparabilité pessimiste ainsi défini ne peut que décroître de façon monotone par simulation inverse :

$$\forall p_1, p_2 \in Q, p_1 \precsim q_1 \wedge p_2 \precsim q_2 \implies \mathsf{septime_P}^*(p_1, p_2) \leq \mathsf{septime_P}^*(q_1, q_2).$$

Afin d'appliquer cette restriction au délai de séparabilité d'un état q, il est nécessaire de l'étendre aux paires séparantes. Un obstacle à cette extension est que des paires séparantes peuvent disparaître ou apparaître par simulation inverse. Les paires séparantes *stables* (Def. 14 p. 46), sous-ensemble des paires séparantes, ne peuvent qu'apparaître. Dans ces deux cas, nous ne pouvons donc pas borner le délai de séparabilité pessimiste d'un état, même en appliquant la restriction montrée ci-dessus. Dans la suite du manuscrit et en particulier lors de l'opération de *raffinement*, la définition de délai de séparabilité pessimiste sera amendée de façon à être relative à un ensemble *fixe* de paires séparantes stables.

2.5 Synthèse

Nous avons dans ce chapitre choisi les machines de Moore comme modèle de nos systèmes synchrones faibles. La bisimilarité a été définie comme relation d'équivalence sur ces machines.

Nous avons ensuite défini la notion de délai de séparabilité, qui permet de raisonner sur le temps (en terme de nombre de transitions) nécessaire à une machine pour traiter une information. Cette notion de délai de séparabilité est fondée d'une part sur la notion classique de dépendance fonctionnelle et d'autre part sur la notion de non bisimilarité. Cette notion de délai de séparabilité a été raffinée en une variante optimiste et une variante pessimiste.

Finalement, nous avons étudié la conservation du délai de séparabilité par les relations de simulation et de simulation inverse, qui seront utilisées pour définir l'abstraction et le raffinement de machines au chapitre suivant. Une première conclusion de cette étude est que l'information contenue dans le caractère branchant d'une machine n'est pas préservée par simulation ni par simulation inverse. Nous avons vu que dans certains cas, il est possible de sur-approximer le délai optimiste d'un état après l'opération de simulation inverse en en considérant une restriction.

Notation	*Description*	*Référence*
$\mathsf{c{-}ex}(p,q)$	Ensemble des contre-exemples à la bisimilarité de p et q.	Prop. 2.7 p. 35
$\mathsf{reactive}(q)$	Propriété de réactivité (*non-constance*) de q	Def. 6 p. 6
$\mathsf{SepPairs}(q)$	Paires séparantes de q.	Def. 6 p. 6
$\mathsf{SepPairs}_*(q)$	Paires séparantes *stables* de q, basées sur l'existence de pseudo-séparateurs pour tous les états atteignables par la paire séparante.	Def. 14 p. 14
$\mathsf{SepEx}(p,q)$	Séquences d'exécution séparante, correspondant à un chemin sur un contre-exemple à la bisimilarité de p et q	Def. 7 p. 36
$\mathsf{S}(p,q)$	Séparateurs de p et q. Mots étiquetant les séquences d'exécution séparantes de p et q.	Def. 8 p. 37
$\mathsf{S}_!(p,q)$	Séparateurs effectifs de p et q. Séparateurs dont toutes les séquences d'exécution sont séparantes.	Def. 8 p. 37
$\mathsf{S}_*(p,q)$	Pseudo-séparateurs de p et q. Mots minimaux induisant des langages de sortie disjoints.	Def. 13 p. 44
$\mathsf{septime}_\mathsf{O}(p,q)$	Délai de séparabilité optimiste de p et q.	Def. 9 p. 9
$\mathsf{septime}_\mathsf{O}(p,q)$	Délai de séparabilité pessimiste de p et q.	Def. 10 p. 10

FIGURE 2.15 – Tableau récapitulatif des principales notations utilisées dans le chapitre.

3

Relation de simulation pour l'abstraction et le raffinement

Nous commençons par donner quelques définitions préliminaires portant sur les ensembles ordonnés. Nous proposons ensuite une méthode de spécification de propriétés fonctionnelles simples par l'utilisation d'*abstractions de données* ayant des structures de treillis. Ces abstractions de données sont utilisées comme alphabet pour une variante des machines de Moore que nous appelons *machines sur treillis*. Nous étudions les conditions sous lesquelles il est possible d'analyser le délai de séparabilité sur ces machines et sous lesquelles il est conservé par concrétisation des données.

Toujours dans le cadre des machines sur treillis, nous étudions le délai de séparabilité dans le cas général, c'est-à-dire entre des *ports* d'entrée et de sortie particuliers – notions que nous définissons également. Enfin, tous ces résultats et ces techniques sont mis en oeuvre dans un formalisme de développement incrémental ou l'opération de raffinement est précisément la relation de simulation des machines sur treillis. Nous insistons sur le fait que l'abstraction de données ne correspond pas à une démarche inverse au raffinement : nous nous servons de cette méthode (issue du domaine de la vérification automatique) pour proposer un formalisme de développement plus riche.

3.1 Rappel de notations sur les ensembles ordonnés et les machines

3.1.1 Ensembles ordonnés

Ordre partiel, pré-ordre, ordre total. Soit A un ensemble quelconque. Toute relation réflexive, transitive et antisymétrique $\preccurlyeq_A \subseteq A \times A$ est une *relation d'ordre* et $\langle A, \preccurlyeq_A \rangle$ est alors appelé *ordre partiel* ou *poset*. Si $\forall x, y \in A, x \preccurlyeq_A y \vee y \preccurlyeq_A x$ alors $\langle A, \preccurlyeq_A \rangle$ est un *ordre total*. Si la relation n'est pas antisymétrique, $\langle A, \preccurlyeq_A \rangle$ est un *pré-ordre*.

Fonctions monotones sur un pré-ordre. Soient $\langle A, \preccurlyeq_A \rangle$ et $\langle B, \preccurlyeq_B \rangle$ deux pré-ordres et soit $f : A \to B$ une fonction totale. Nous dirons que f est monotone si elle satisfait la propriété :

$$x \preccurlyeq_A y \implies f(x) \preccurlyeq_B f(y).$$

Ordre produit. Si $\langle A, \preccurlyeq_A \rangle$ et $\langle B, \preccurlyeq_B \rangle$ sont deux ordres partiels, l'ensemble $A \times B$ peut être ordonné par l'*ordre produit* $\preccurlyeq_{A\times B}$ défini comme :

$$(a_1, b_1) \preccurlyeq_{A\times B} (a_2, b_2) \iff a_1 \preccurlyeq_A a_2 \wedge b_1 \preccurlyeq_B b_2.$$

Cette opération est généralisable à tout produit fini de posets. En particulier si $\langle A, \preccurlyeq_A \rangle$ est un poset, l'ensemble des mots A^* est muni d'un ordre produit pour les mots de même longueur : $\forall w_1 w_2 \in A^n, w_1 \preccurlyeq_{A^*} w_2 \iff \forall i, w_1[i] \preccurlyeq_A w_2[i]$.

Treillis. Un *treillis* est un ordre partiel $\langle A, \preccurlyeq_A \rangle$ disposant d'une opération de borne supérieure $\sqcup_A \in A \times A \to A$ et d'une opération de borne inférieure $\sqcap_A \in A \times A \to A$ commutatives et associatives. Dans un treillis, nous avons l'équivalence $a \preccurlyeq_A b \iff a \sqcup_A b = b$. La relation d'ordre $\preccurlyeq_A$ peut donc être déduite de $\sqcup_A$ ou $\sqcap_A$. Un treillis est *borné* si il est doté d'un élément minimal noté $\bot$ et d'un élément maximal noté $\top$. Une telle structure est notée $\langle A, \preccurlyeq_A, \sqcup_A, \sqcap_A, \top_A, \bot_A \rangle$.

Construction de treillis particuliers. Une construction importante est celle qui à tout ensemble A associe le treillis des parties de A. Cette construction sera notée $\mathcal{T}_{\wp(A)}$:

$$\mathcal{T}_{\wp(A)} = \langle \wp(A), \subseteq, \cup, \cap, A, \varnothing \rangle.$$

Afin de modéliser les types de données, nous aurons également besoin de définir l'extension aux treillis du produit cartésien, de façon similaire à l'ordre produit. Soient deux treillis :

$$\mathcal{T}_A = \langle A, \preccurlyeq_A, \sqcup_A, \sqcap_A, \top_A, \bot_A \rangle, \quad \mathcal{T}_B = \langle B, \preccurlyeq_B, \sqcup_B, \sqcap_B, \top_B, \bot_B \rangle.$$

Le treillis produit est $\mathcal{T}_{A\times B} = \langle A \times B, \preccurlyeq_{A\times B}, \sqcup_{A\times B}, \sqcap_{A\times B}, (\top_A, \top_B), (\bot_A, \bot_B) \rangle$ où :

$$\begin{array}{rcl} \sqcup_{A\times B} ((a_1, b_1), (a_2, b_2)) & = & (a_1 \sqcup_A a_2,\ b_1 \sqcup_B b_2) \\ \sqcap_{A\times B} ((a_1, b_1), (a_2, b_2)) & = & (a_1 \sqcap_A a_2,\ b_1 \sqcap_B b_2) \end{array}$$

Cette opération est généralisable à tout produit fini de treillis, et en particulier aux *mots* de même longueur. Soit $\mathcal{T}_A$ un treillis tel que défini précédemment. Le treillis correspondant aux mots de longueur n est :

$$\mathcal{T}_A^n = \langle A^n, \preccurlyeq_A^n, \sqcup_A^n, \sqcap_A^n, \langle \bot_A, \ldots, \bot_A \rangle, \langle \top_A, \ldots, \top_A \rangle, \rangle,$$

dont les composantes sont définies comme suit :

$$\begin{array}{rcl} w_1 \preccurlyeq_A^n w_2 & \iff & \exists n, \forall i < n, w_1[i] = w_2[i] \wedge w_1[n] \preccurlyeq_A w_2[n], \\ w_1 \sqcup_A^n w_2 & = & \lambda i. w_1[i] \sqcup_A w_2[i], \\ w_1 \sqcap_A^n w_2 & = & \lambda i. w_1[i] \sqcap_A w_2[i]. \end{array}$$

Il existe un tel treillis pour toute longueur $n \in \mathbb{N}$. Nous ferons référence à cette collection de treillis par $\mathcal{T}^*$.

3.1.2 Extensions syntaxiques

Nous procédons à la définition d'une extension purement syntaxique des machines de Moore permettant l'utilisation de variables, d'assignements et de déréférencements. Cette extension n'augmente pas l'expressivité de notre formalisme et est orthogonale aux considérations sur l'abstraction traitées dans ce chapitre.

3.1.2.1 Environnements

Soit $\mathcal{V}$ un ensemble dénombrable de noms de variables. Si $V \subset \mathcal{V}$ est un ensemble *fini* de variables, un *environnement* $\Gamma_V = \{(v_0, a_0), (v_1, a_1), \ldots\}$ est une relation fonctionnelle associant à chaque variable $v_i \in V$ un élément a_i d'un ensemble E_{v_i}. Formellement, $\Gamma_V \in \prod_{v \in V} E_v$. Si $x \in V$, nous notons $\Gamma_V(x)$ la valeur associée à la variable x.

3.1.2.2 Affectation de variable

Soit Γ_V un environnement, $v \in V$ une variable et x une donnée. Nous notons $[\Gamma_V \mid x \mapsto a]$ la mise à jour de la variable x par la valeur a dans Γ_V. Cette construction vérifie la propriété suivante :

$$[\Gamma_V \mid x \mapsto a](x) = a \quad \wedge \quad \forall x' \neq x \in V, [\Gamma_V \mid x \mapsto a](x') = \Gamma_V(x').$$

3.1.2.3 Machines avec variables

Soit In un alphabet *d'entrée* et Out un alphabet de *sortie*. Une machine de Moore avec variables M sur In et Out est la donnée de :

- un ensemble fini d'états Q
- un ensemble fini de variables $V \in \mathcal{V}$,
- un ensemble d'arcs $E \subseteq Q \times In \times Q$,
- une fonction de sortie $\mathsf{out} : Q \times \prod_{v \in V} E_v \to Out$,
- une fonction $\mathsf{transfer} : E \times \prod_{v \in V} E_v \to \prod_{v \in V} E_v$,
- un environnement initial $\Gamma_V^0 \in \prod_{v \in V} E_v$,
- un état initial $q_0 \in Q$.

La fonction de transfert associée à une séquence d'exécution est la composition des fonctions de transfert correspondant aux transitions empruntées. Le mot en sortie est calculé conjointement à partir de l'état et de l'environnement en chaque point. Étant donné un mot $w \in In^*$, une séquence d'exécution est un mot de la forme :

$$(q_0, \Gamma_V^0).w[0].(q_1, \Gamma_V^1).w[1].(q_2, \Gamma_V^2).w[2]\ldots$$

Avec :

$$\begin{aligned} \Gamma_V^1 &= \mathsf{transfer}((q_0, w[0], q_1), \Gamma_V^0) \\ \Gamma_V^2 &= \mathsf{transfer}((q_1, w[1], q_2), \Gamma_V^1) \\ &\cdots \\ \Gamma_V^{i+1} &= \mathsf{transfer}((q_i, w[i], q_{i+1}), \Gamma_V^i) \end{aligned}$$

Dans les exemples utilisant les machines avec variables, les fonctions de transfert associées aux arcs pourront être données comme des suites d'instructions dans un pseudo-langage algorithmique.

3.2 Abstraction

La puissance de calcul est une ressource bornée, et la spécification autant que la vérification automatique ont pour frontière la taille excessive de l'espace d'états du système en cours de réalisation. Une solution classique est d'approximer le système en question. Cette perte d'information entraîne une précision moindre des analyses automatiques. En contrepartie, il devient possible de vérifier en temps raisonnable ce qui n'était auparavant pas envisageable, et

une vision "globale" du système est permise. Ceci est d'une importance particulière lors d'une démarche de développement par raffinement, partant d'une version "abstraite" du système et arrivant à une version concrète, composée de machines de Moore déterministes.

Cette méthode par approximation n'a de sens que si nous sommes capables de montrer que les propriétés d'intérêt sont bien préservées par l'abstraction (l'analyse doit rester *correcte*). Un cadre formel général pour produire des analyses correctes par abstraction a été proposé par Cousot et Cousot [29]. Une adaptation au cadre des systèmes concurrents a été par la suite proposée dans [49]. Nous proposons une application de cette technique à l'analyse du délai de séparabilité, selon le plan exposé ci-dessous.

- Nous exposons brièvement le concept mathématique de correspondance de Galois (Sec. 3.2.1), permettant la définition des fonctions d'*abstraction* et de *concrétisation* entre préordres (et donc entre treillis).
- En guise de première application des correspondance de Galois, nous introduisons la notion d'*abstraction de type de donnée*, permettant d'approximer les alphabets des machines (Sec. 3.2.2 p. 55).
- Nous proposons les *machines sur treillis*, une extension des machines de Moore non-déterministes leur permettant de travailler sur les abstractions de types de données (Sec. 3.2.3 p. 60). Nous définissons une relation de simulation adaptée à ces machines. Une fonction de *concrétisation* des machines sur treillis vers les machines concrètes (définies au chapitre 2) est donnée.
- Enfin, nous prouvons la *correction* de l'analyse du délai de séparabilité sur machines sur treillis (Sec. 3.2.4 p. 65) : effectuer l'analyse du délai de séparabilité sur une machine sur treillis sur-approxime le délai de séparabilité des machines concrètes correspondantes.

3.2.1 Abstraction et correspondances de Galois

Les méthodes d'abstraction procèdent généralement en trois étapes. Initialement, on définit une abstraction sur les *données* par une fonction associant à chaque valeur concrète une valeur abstraite correspondante. Par la suite, une sémantique opérationnelle prenant en compte l'incertitude induite par l'abstraction est donnée pour le modèle de calcul considéré. Ceci correspond à donner une abstraction de l'espace d'états. Finalement, il est prouvé que cette nouvelle sémantique préserve les propriétés d'intérêt. Les relations entre données concrètes et abstraites d'une part et espace d'états concret et abstrait d'autre part sont classiquement données comme des correspondances de Galois [33]. La preuve de correction de l'abstraction en est grandement facilitée.

Soit S un ensemble quelconque. Une notion naturelle *d'approximation* d'éléments de S est de considérer les parties de S. Ainsi, une partie $P = \{s_1, s_2, \ldots\} \subseteq S$ correspond à une valeur non-déterministe inconnue $s_i \in P$. Mais en pratique, manipuler des parties d'ensembles est calculatoirement coûteux. Dans certains cas, il existe une ou plusieurs représentations *abstraites* de l'ensemble des parties $\wp(S)$, munies de la même structure de treillis mais disposant d'opérations efficaces, au prix d'une perte de précision. Lorsqu'il est possible de montrer qu'à toute partie $P \subseteq S$ on peut associer une *meilleure abstraction*, il existe une correspondance de Galois [33] entre l'ensemble des parties concret et l'abstraction.

Définition 15 (Correspondance de Galois)**.** *Soient $\mathcal{P}_C = \langle C, \leqslant_C \rangle$ et $\mathcal{P}_A = \langle A, \leqslant_A \rangle$ deux pré-ordres. Une correspondance de Galois entre $\mathcal{P}_C$ et $\mathcal{P}_A$ est un couple de fonctions monotones $\alpha : C \to A$ et $\gamma : A \to C$ telles que :*

$$\forall c \in C, \forall a \in A, \alpha(c) \leqslant_A a \iff c \leqslant_C \gamma(a).$$

*La fonction α est la fonction d'*abstraction *et la fonction γ est celle de* concrétisation*. Nous écrirons $(\alpha, \gamma) \in Galois(\mathcal{P}_C, \mathcal{P}_A)$ pour noter une telle correspondance. Cette situation est représentée en Fig. 3.1. Noter que ces définitions s'étendent directement au cas des ordres partiels et donc des treillis.*

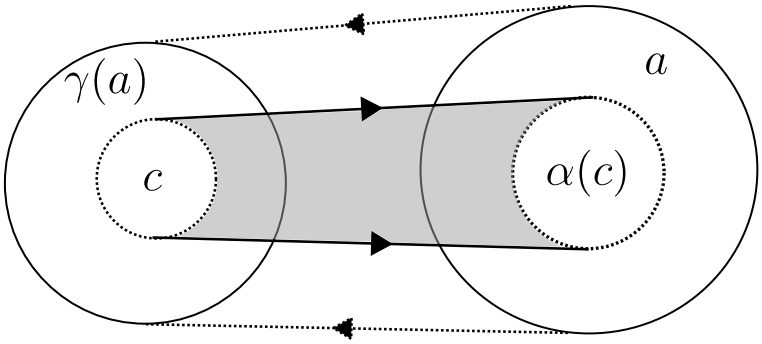


FIGURE 3.1 – Correspondance de Galois

Les correspondances de Galois sont fermées par composition, grâce à la préservation de la monotonie des fonctions sous-jacentes par composition. Il existe également un élément neutre, ce qui dote l'ensemble des correspondances de Galois d'une structure de monoïde.

Définition 16 (Monoïde des correspondances de Galois)**.** *Soient $(\alpha_1, \gamma_1) \in Galois(\mathcal{P}_A, \mathcal{P}_B)$ et $(\alpha_2, \gamma_2) \in Galois(\mathcal{P}_B, \mathcal{P}_C)$ deux correspondances de Galois. Alors nous avons :*

$$(\alpha_2 \circ \alpha_1, \gamma_1 \circ \gamma_2) \in Galois(\mathcal{P}_A, \mathcal{P}_C),$$

et l'associativité de cette opération provient directement de l'associativité de la composition de fonctions. L'élément neutre est la famille de correspondances $(id, id) \in Galois_{X,X}$ pour tout pré-ordre $\langle X, \leqslant_X \rangle$.

3.2.2 Abstraction de type de donnée

L'ensemble des types de données concrets est défini comme un monoïde ayant pour générateurs les types de données de $Basic$ (cf. Sec. 2.1.2 p. 24). Nous allons définir l'ensemble des types de donnée abstraits de façon similaire, en associant à chaque type de donnée basique une abstraction sous forme d'un treillis. Nous procéderons alors à la définition d'un ensemble de types de données comme le monoïde ayant pour générateurs ces treillis et fermé par produit de treillis.

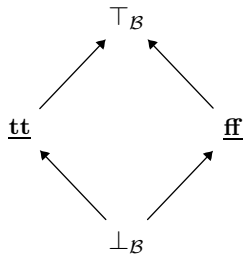
Afin d'illustrer les définitions concernant les correspondances de Galois et d'initier le programme énoncé ci-dessus, nous commençons par introduire quelques exemples classiques d'abstractions de types de données.

3.2.2.1 Exemple d'abstraction sur les booléens

Soit **bool** = $\{\mathbf{tt}\,;\mathbf{ff}\}$ le type des booléens. Soit $\mathcal{T}_{\wp(\mathbf{bool})} = \langle \wp(\mathbf{bool}), \subseteq, \cup, \cap, \mathbf{bool}, \varnothing\rangle$ le treillis des parties associé à **bool**. Soit $\mathcal{B} = \{\underline{\mathbf{tt}}\,;\underline{\mathbf{ff}}\,;\top_{\mathcal{B}};\bot_{\mathcal{B}}\}$ un ensemble ordonné par la relation $\preccurlyeq_{\mathcal{B}}$:

$$\forall x, x \preccurlyeq_{\mathcal{B}} \top_{\mathcal{B}} \wedge \bot_{\mathcal{B}} \preccurlyeq_{\mathcal{B}} x.$$

Cette relation d'ordre est illustrée par le diagramme qui suit, où seule la partie "non-transitive" de la relation est donnée.



Les fonctions d'abstraction α et de concrétisation γ sont définies ci-dessous :

$$\begin{array}{lcl|lcl} \alpha_{\mathcal{B}}(\varnothing) & = & \bot_{\mathcal{B}} & \gamma_{\mathcal{B}}(\bot_{\mathcal{B}}) & = & \varnothing \\ \alpha_{\mathcal{B}}(\mathbf{bool}) & = & \top_{\mathcal{B}} & \gamma_{\mathcal{B}}(\top_{\mathcal{B}}) & = & \mathbf{bool} \\ \alpha_{\mathcal{B}}(\{b\}) & = & \underline{b} & \gamma_{\mathcal{B}}(\underline{b}) & = & b \ \ (b \in \{\mathbf{tt}\,,\mathbf{ff}\}) \end{array}$$

Nous déduisons de la relation d'ordre les opérations de bornes supérieure et inférieure :

$$\begin{array}{lcl|lcl} \bot_{\mathcal{B}} \sqcup_{\mathcal{B}} x & = & x \sqcup_{\mathcal{B}} \bot_{\mathcal{B}} = x & \top_{\mathcal{B}} \sqcap_{\mathcal{B}} x & = & x \sqcap_{\mathcal{B}} \top_{\mathcal{B}} = x \\ \top_{\mathcal{B}} \sqcup_{\mathcal{B}} x & = & x \sqcup_{\mathcal{B}} \top_{\mathcal{B}} = \top_{\mathcal{B}} & \bot_{\mathcal{B}} \sqcap_{\mathcal{B}} x & = & x \sqcap_{\mathcal{B}} \bot_{\mathcal{B}} = \bot_{\mathcal{B}} \\ \underline{b_1} \sqcup_{\mathcal{B}} \underline{b_2} & = & \top_{\mathcal{B}} & \underline{b_1} \sqcap_{\mathcal{B}} \underline{b_2} & = & \bot_{\mathcal{B}} \qquad b_i \in \mathbf{bool} \end{array}$$

Soit $\mathcal{T}_{\mathcal{B}} = \langle \mathcal{B}, \preccurlyeq_{\mathcal{B}}, \sqcup_{\mathcal{B}}, \sqcap_{\mathcal{B}}, \top_{\mathcal{B}}, \bot_{\mathcal{B}}\rangle$ le treillis résultant. Le lemme suivant exprime le fait que ce treillis constitue bien une abstraction des booléens.

Lemme 3.1. *La propriété suivante est vérifiée :* $(\alpha_{\mathcal{B}}, \gamma_{\mathcal{B}}) \in Galois(\mathcal{T}_{\wp(\mathbf{bool})}, \mathcal{T}_{\mathcal{B}})$.

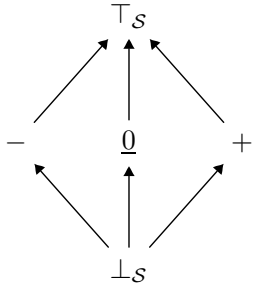
Démonstration. Il suffit de montrer $X \subseteq \gamma_{\mathcal{B}}(\alpha_{\mathcal{B}}(X))$ pour toute partie $X \in \wp(\mathbf{bool})$. Nous avons $\wp(\mathbf{bool}) = \{\varnothing;\ \{\mathbf{tt}\};\ \{\mathbf{ff}\};\ \{\mathbf{tt}\,;\mathbf{ff}\}\}$. Le résultat suit par application directe des définitions des fonctions de concrétisation et d'abstraction dans chaque cas. □

3.2.2.2 Exemple d'abstraction sur les entiers : les signes

Soit **int** le type des entiers finis. Soit $\mathcal{S} = \{+, -, \underline{0}, \top_{\mathcal{S}}, \bot_{\mathcal{S}}\}$ l'ensemble de signes (positif, négatif, nul, signe inconnu, aucun signe). Cet ensemble est ordonné par la relation $\preccurlyeq_{\mathcal{S}}$:

$$\forall x, x \preccurlyeq_{\mathcal{S}} \top_{\mathcal{S}} \wedge \ \bot_{\mathcal{S}} \preccurlyeq_{\mathcal{S}} x.$$

Cette relation d'ordre est décrite par le diagramme qui suit.



La correspondance de Galois est définie entre l'ensemble des parties (finies) d'entiers et les signes. Les fonctions d'abstraction et de concrétisation sont $\alpha_{\mathcal{S}}$ et $\gamma_{\mathcal{S}}$, définies ainsi :

$$
\begin{array}{llll|ll}
\alpha_{\mathcal{S}}(X) & = + & \Longleftrightarrow & X \subseteq [1; max\_int] & \gamma_{\mathcal{S}}(\top_{\mathcal{S}}) & = \textbf{int} \\
\alpha_{\mathcal{S}}(\{0\}) & = \underline{0} & & & \gamma_{\mathcal{S}}(\bot_{\mathcal{S}}) & = \varnothing \\
\alpha_{\mathcal{S}}(X) & = - & \Longleftrightarrow & X \subseteq [min\_int; -1] & \gamma_{\mathcal{S}}(+) & = [1; max\_int] \\
\alpha_{\mathcal{S}}(\varnothing) & = \bot_{\mathcal{S}} & & & \gamma_{\mathcal{S}}(-) & = [min\_int; -1] \\
\alpha_{\mathcal{S}}(X) & = \top_{\mathcal{S}} & & \text{si autre cas.} & \gamma_{\mathcal{S}}(\underline{0}) & = \{0\}
\end{array}
$$

Les opérations de bornes supérieure et inférieure $\sqcup_{\mathcal{S}}$ et $\sqcap_{\mathcal{S}}$ peuvent être déduites directement de la relation d'ordre $\preccurlyeq_{\mathcal{S}}$. Soit le treillis $\mathcal{T}_{\mathcal{S}} = \langle \mathcal{S}, \preccurlyeq_{\mathcal{S}}, \sqcup_{\mathcal{S}}, \sqcap_{\mathcal{S}}, \top_{\mathcal{S}}, \bot_{\mathcal{S}} \rangle$.

Lemme 3.2. *La propriété suivante est vérifiée :* $(\alpha_{\mathcal{S}}, \gamma_{\mathcal{S}}) \in Galois(\mathcal{T}_{\wp(\textbf{\textit{int}})}, \mathcal{T}_{\mathcal{S}})$.

Démonstration. Soit $X \subseteq$ **int** un ensemble d'entiers finis. La preuve de $X \subseteq \gamma_{\mathcal{S}}(\alpha_{\mathcal{S}}(X))$ procède par analyse par cas. Nous montrons l'inclusion dans un de ces cas, les autres étant similaires. Si $\alpha_{\mathcal{S}}(X) = +$, nous avons nécessairement $X \subseteq [1; max\_int]$. Or, $\gamma_{\mathcal{S}}(+) = [1; max\_int]$. Par transitivité, $X \subseteq \gamma_{\mathcal{S}}(\alpha_{\mathcal{S}}(X))$. □

3.2.2.3 Exemple d'abstraction sur les réels : les intervalles

Nous donnons un dernier exemple d'abstraction sur un ensemble qui n'est pas un type de donnée. Considérons l'ensemble des nombres réels $\mathbb{R}$ et celui des nombres rationnels $\mathbb{Q}$. Tout nombre réel $r \in \mathbb{R}$ peut être encadré par deux nombres rationnels $n_0, n_1 \in \mathbb{Q}$ tels que $n_0 \leq r \leq n_1$. Soit $\mathcal{I} = \{\, [x; y] \in \mathbb{Q} \times \mathbb{Q} \mid x \leq y \} \cup \{\bot_{\mathcal{I}}; \top_{\mathcal{I}}\}$ l'ensemble des intervalles sur les rationnels étendu par un élément minimal $\bot_{\mathcal{I}}$ et un élément maximal $\top_{\mathcal{I}}$. Les opérations du treillis associé sont définies ci-dessous :

$$
\begin{array}{lcl}
[a; b] \sqcup_{\mathcal{I}} [c; d] & = & [min(a, c);\ max(b, d)]; \\
[a; b] \sqcap_{\mathcal{I}} [c; d] & = & [max(a, c);\ min(b, d)]; \\
[a; b] \preccurlyeq_{\mathcal{I}} [c; d] & = & c \leq a \wedge b \leq d; \\
\forall x, \bot_{\mathcal{I}} \preccurlyeq_{\mathcal{I}} x; & & \\
\forall x, x \preccurlyeq_{\mathcal{I}} \top_{\mathcal{I}}. & &
\end{array}
$$

La correspondance de Galois est définie entre l'ensemble des parties finies de réels et les intervalles. Un exemple de fonction d'abstraction est le suivant, pour tout $X \subset \mathbb{R}$:

$$\alpha_{\mathcal{I}}(X) = [\, \lfloor min(X) \rfloor; \lceil max(X) \rceil \,].$$

Cette abstraction encadre la partie X par la partie entière inférieure de son minimum et la partie entière supérieure de son maximum. Bien entendu, des fonctions plus précises peuvent être imaginées. La fonction de concrétisation associée est $\gamma_{\mathcal{I}}([a;b]) = \{r \in \mathbb{R} \mid a \leq i \leq b\}$. Le treillis résultant est le suivant : $\mathcal{T}_{\mathcal{I}} = \langle \mathcal{I}, \preccurlyeq_{\mathcal{I}}, \sqcup_{\mathcal{I}}, \sqcap_{\mathcal{I}}, \top_{\mathcal{I}}, \bot_{\mathcal{I}} \rangle$.

Lemme 3.3. *La propriété suivante est vérifiée :* $(\alpha_{\mathcal{I}}, \gamma_{\mathcal{I}}) \in Galois(\mathcal{T}_{\wp_{fin}(\mathbb{R})}, \mathcal{T}_{\mathcal{I}})$.

Démonstration. La preuve est similaire au cas des abstractions précédentes et est omise. □

3.2.2.4 Monoïde des abstractions de types de données

Les abstractions montrées précédemment concernent des types de données atomiques (entiers, booléens). Nous définissons ici les abstractions de types de données de $\mathfrak{D}$ par utilisation du produit de treillis sur les abstractions correspondant à chaque type de donnée atomique.

Soit $C = D_1 \times D_2 \times \ldots \times D_n \in \mathfrak{D}$ un type de donnée concret tel que $D_i \in Basic$. A chaque D_i, nous pouvons associer son treillis des parties $\mathcal{T}_{\wp(D_i)}$. Soit une collection de treillis $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_n$ tels que :

$$\mathcal{T}_i = \langle A_i, \preccurlyeq_i, \sqcup_i, \sqcap_i, \top_i, \bot_i \rangle.$$

Supposons de surcroît que ces treillis soient en correspondance de Galois avec les types de données concrets composante par composante :

$$\forall i \in [1;n], \exists (\alpha_i, \gamma_i) \in Galois(\mathcal{T}_{\wp(D_i)}, \mathcal{T}_i).$$

Nous pouvons alors construire une *abstraction* du type de donnée $C = D_1 \times D_2 \times \ldots \times D_n$, comme montré par le lemme suivant.

Lemme 3.4 (Abstractions de types de données)**.** *Soient les fonctions :*

$$\alpha \ : \ \wp(C) \to A_1 \times A_2 \times \ldots \times A_n, \qquad \gamma \ : \ A_1 \times A_2 \times \ldots \times A_n \to \wp(C),$$

définies par les équations suivantes :

$$\begin{array}{lcl} \alpha(X) & = & (\alpha_1(\pi_1(X)),\ \alpha_2(\pi_2(X)), \ldots, \alpha_n(\pi_n(X)))\,, \\ \gamma(a_1, \ldots, a_n) & = & \{(d_1,\ \ldots,\ d_n) \mid \forall i,\ d_i \in \gamma_i(a_i)\}\,. \end{array}$$

Ces fonctions forment une correspondance de Galois :

$$(\alpha, \gamma) \in Galois(\mathcal{T}_{\wp(C)}, \mathcal{T}_1 \times \ldots \times \mathcal{T}_n).$$

Démonstration. Observons tout d'abord que ces fonctions sont monotones. Considérons les ensembles X et Y tels que $X \subseteq Y \subseteq \wp(C)$. Par définition de la relation de l'ordre produit, nous avons $\alpha(X) \preccurlyeq \alpha(Y)$ si $\alpha_i(\pi_i(X)) \preccurlyeq \alpha_i(\pi_i(Y))$. Cette dernière propriété découle directement de la monotonie des fonctions α_i. La preuve de la monotonie de γ suit le même principe.

Soit $X \subseteq \wp(C)$ et montrons $X \subseteq \gamma(\alpha(X))$. Après application des définitions et simplification, le but se traduit en :

$$X \subseteq \{(d_1,\ \ldots,\ d_n) \mid \forall i,\ d_i \in \gamma_i(\alpha_i(\pi_i(X)))\}\,.$$

Or, l'inclusion $\pi_i(X) \subseteq \gamma_i(\alpha_i(\pi_i(X)))$ suit par propriété de la correspondance de Galois (α_i, γ_i). Ceci nous permet de conclure la preuve par application de la définition du produit cartésien. □

Ce lemme justifie la terminologie suivante : nous appellerons tout treillis $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_n$ satisfaisant les conditions édictées ci-dessus une *abstraction de type de donnée*. Qui plus est, nous appellerons *correspondances de Galois concrètes* les couples $(\alpha_i, \gamma_i) \in Galois(\mathcal{T}_{\wp(D_i)}, \mathcal{T}_i)$ liant les abstractions de types de données aux types concrets.

Exemple 3.5 (Correspondances de Galois concrètes) **:** L'abstraction $\mathcal{T}_\mathcal{B}$ est en correspondance de Galois concrète avec $\mathcal{T}_{\wp(\textbf{bool})}$. Les abstractions $\mathcal{T}_\mathcal{S}$ (signes) et $\mathcal{T}_\mathcal{I}$ (intervalles) sont en correspondance de Galois concrète avec $\mathcal{T}_{\wp(\textbf{int})}$.

3.2.2.5 Abstraction de langages

Nous avons vu en Sec. 2.4.2.2 p. 43 que certains séparateurs peuvent être caractérisés par le fait que leurs langages de sortie sont disjoints. Ceci motive quelques définitions sur les langages dont les symboles sont tirés d'abstractions de types de données. Nous définissons successivement des extensions abstraites des mots puis des langages et enfin l'opération d'intersection sur langages abstraits. Dans ce qui suit, nous considérons une abstraction de type de donnée $\mathcal{T} = \langle A, \leqslant, \sqcup, \sqcap, \top, \bot \rangle$. Puisque $\mathcal{T}$ est une abstraction de type de donnée, il existe une correspondance de Galois (α, γ) entre $\mathcal{T}$ et l'ensemble des parties d'un type de donnée concret C :

$$(\alpha, \gamma) \in Galois(\mathcal{T}_{\wp(C)}, \mathcal{T}).$$

Nous pouvons l'étendre en une correspondance entre le treillis des mots $\mathcal{T}^*$ et le treillis des parties de mots concrets de longueur n. Le lemme qui suit exprime cet état de fait.

Lemme 3.6 (Abstraction de mot)**.** *Soient les fonctions de concrétisation :*

$$\gamma^*(w) \quad = \quad \{v \mid \forall i \in [0; n-1], v[i] \in \gamma(w[i])\},$$

et d'abstraction :

$$\alpha^*(X) \quad = \quad \alpha(X_0).\alpha(X_1)\ldots\alpha(X_{n-1}), \textit{où } X_i = \{v[i] \mid v \in X\}.$$

Ces fonctions forment une correspondance de Galois :

$$(\alpha^*, \gamma^*) \in Galois(\mathcal{T}_{\wp(C^*)}, \mathcal{T}^*).$$

Démonstration. La preuve est similaire à celle du lemme 3.4. □

Nous appellerons toute séquence $w \in A^*$ (cf. p. 52) est une *abstraction de mot* sur l'alphabet abstrait A.

Exemple 3.7 (Abstraction de mot) **:** Considérons l'alphabet abstrait des signes $\mathcal{T}_\mathcal{S} \in \mathfrak{D}^\alpha$. Soit $w = +.-.+$ un mot sur cet alphabet. Nous avons :

$$\gamma^*(w) = \{p_1.n.p_2 \mid \exists p_1, p_2 \in [1;\ max\_int] \ \wedge\ \exists n \in [min\_int; -1]\}.$$

Noter que si une abstraction de mot w contient le symbole $\bot$ à un indice quelconque, nous obtenons par concrétisation $\gamma^*(w) = \varnothing$. Dans la suite, nous considérerons les abstractions de langages équivalentes modulo l'existence de tels mots.

A partir de la notion d'abstraction de mot il semble naturel de définir les abstractions de langage comme des ensembles de ces mots. Un ensemble d'abstractions de mots sera par exemple obtenu après lecture d'un mot par une machine de Moore sur treillis non-déterministe (cf. Sec. 3.2.3 p. 60). Cependant, les opérations ensemblistes sur ces ensembles d'abstractions de mots n'ont plus leur sens usuel. Nous avons en particulier besoin de définir l'intersection d'abstractions de langages.

Définition 17 (Intersection d'abstractions de langages)**.** *Soient $L_1 \subseteq A^*$ et $L_2 \subseteq A^*$ deux abstractions de langages. Leur intersection abstraite est calculée comme suit :*

$$L_1 \overline{\bigcap} L_2 = \{w \mid w = w_1 \sqcap^* w_2, \exists w_1 \in L_1, w_2 \in L_2\},$$

où $\sqcap^*$ est l'extension de $\sqcap$ à l'ordre lexical (cf. Sec. 3.1.1 p. 52).

Exemple 3.8 : Considérons l' abstraction des entiers par leurs signes $\mathcal{T}_\mathcal{S}$. Soit $L_1 = \{+.\top_\mathcal{S}^*; \underline{0}.\top_\mathcal{S}^*\}$ et soit $L_2 = \{-.\underline{0}^*; +.\underline{0}^*\}$. L_1 est le langage des mots dont le premier élément est positif ou nul, et L_2 est celui des mots dont le premier symbole est non-nul et dont les autres symboles sont nuls. Nous obtenons :

$$\begin{array}{rcl} L_1 \overline{\bigcap} L_2 & = & \{(+.\top_\mathcal{S}^*) \sqcap_\mathcal{S}^* (-.\underline{0}^*);\ (\underline{0}.\top_\mathcal{S}^*) \sqcap_\mathcal{S}^* (-.\underline{0}^*); \\ & & \ (+.\top_\mathcal{S}^*) \sqcap_\mathcal{S}^* (+.\underline{0}^*);\ (\underline{0}.\top_\mathcal{S}^*) \sqcap_\mathcal{S}^* (+.\underline{0}^*)\} \\ & = & \{\bot_\mathcal{S}.\underline{0}^*;\ +.\underline{0}^*;\ \underline{0}.\underline{0}^*\} \\ & \simeq & \{+.\underline{0}^*;\ \underline{0}.\underline{0}^*\} \end{array}$$

3.2.3 Abstraction de l'espace d'état de machines de Moore

Cette section propose une notion d'abstraction de machine construite comme une relation de simulation paramétrée par une abstraction de type de donnée. La définition des machines abstraites et la correspondance de Galois associée doivent être pesées avec soin : d'elles dépendent les propriétés qui seront conservées par abstraction. Nous commençons par définir les machines de Moore sur treillis et les différentes notions qui s'y rapportent.

3.2.3.1 Machines de Moore sur treillis

Une machine de Moore sur treillis est structurellement identique à une machine de Moore à l'exception près que les alphabets d'entrée et de sortie ont une structure de treillis. Posons $\mathcal{T}_{In}, \mathcal{T}_{Out} \in \mathfrak{D}^\alpha$.

$$\begin{array}{rcll} \mathcal{T}_{In} & = & \langle In, \preccurlyeq_I, \sqcup_I, \sqcap_I, \top_I, \bot_I \rangle & \textit{(entrées)} \\ \mathcal{T}_{Out} & = & \langle Out, \preccurlyeq_o, \sqcup_o, \sqcap_o, \top_o, \bot_o \rangle & \textit{(sorties)} \end{array}$$

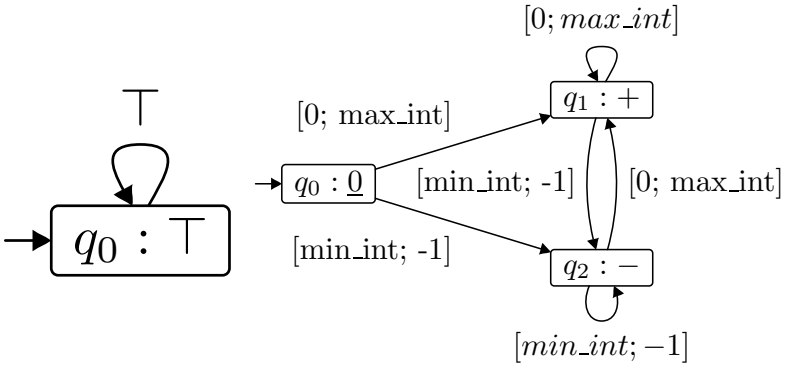
Définition 18 (Machine de Moore sur treillis)**.** *Soit $M = \langle In, Out, Q, \mathsf{out}, E, Q_0 \rangle$ une machine de Moore dont les alphabets d'entrée et de sortie ont des structures de treillis. M est une machine de Moore sur treillis si l'ensemble des arcs E répond à la contrainte de totalité sur treillis, donnée ci-après :*

$$\forall q \in Q, \bigsqcup \{a \mid \exists q \xrightarrow{a} q' \in E\} = \top_I.$$

L'ensemble des machines sur les treillis $\mathcal{T}_{In}$ et $\mathcal{T}_{Out}$ est noté par extension $\mathsf{Moore}(\mathcal{T}_{In}, \mathcal{T}_{Out})$.

Exemple 3.9 : La figure 3.2 montre quelques exemples de machines de Moore sur treillis. La première, en Fig. 3.2(a), montre la machine la plus simple. Réduite à un unique état, elle produit pour toute séquence d'entrée potentiellement n'importe quelle séquence de sortie. La seconde, en Fig. 3.2(b), illustre l'intérêt des abstractions de types de données pour ce qui est de la représentation compacte d'ensembles de transitions de cardinaux importants. Cette machine associe à chaque entrée positive ou nulle le signe "+", et à chaque entrée négative le signe "-".

Le non-déterminisme introduit par la structure de treillis des données impose d'adapter la notion de séquence d'exécution. Un symbole en entrée sera accepté par un arc si et seulement si son "intersection" (i.e. sa borne inférieure) avec l'étiquette de l'arc considéré est non-vide.



(a) Machine triviale (b) Machine calculant le signe

FIGURE 3.2 – Exemples de machines sur treillis

Définition 19 (Séquence d'exécution abstraite)**.** *Nous définissons les séquences d'exécution d'une machine sur treillis* $M = \langle In, Out, Q, \mathsf{out}, E, q_0 \rangle$ *associée à un mot* $w \in In^*$ *comme les séquences appartenant à l'ensemble* $\mathsf{Ex}^\alpha{}_S(q_0, w) \subseteq Q \times (In \times Q)^*$*, avec :*

$$\mathsf{Ex}^\alpha{}_S(q_0, w) = \{q_0.a_0.q_1.a_1.q_2 \ldots \mid \forall i \in \mathbb{N}, w[i] \sqcap_I a_i \neq \bot_I \ \wedge \exists q_i \xrightarrow{a_i} q_{i+1} \in E\}.$$

Le langage de sortie est :

$$\mathcal{L}^*(q_0, w) = \{\mathsf{out}(q_0).\mathsf{out}(q_1).\mathsf{out}(q_2) \ldots \mid q_0.a_0.q_1.a_1.q_2 \ldots \in \mathsf{Ex}^\alpha(q_0, w)\}.$$

Nous pouvons généraliser cette définition à un ensemble de mots $L \subseteq In^*$ *en entrée :*

$$\mathcal{L}^*(q_0, L) = \bigcup_{w \in L} \mathcal{L}^*(q_0, w).$$

3.2.3.2 Relation de simulation pour les machines sur treillis

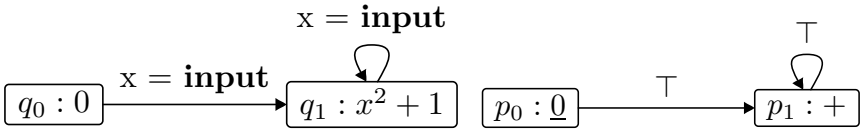
Au fil d'un processus de développement incrémental, le besoin du concepteur en termes de langages de spécifications peut varier. En particulier, l'expression de contraintes plus fines entraîne l'utilisation de langages plus expressifs. Les abstractions sont liées entre elles par une relation d'ordre correspondant à leurs *précisions* relatives, et nous exploitons cette structure afin de permettre l'utilisation d'abstractions de plus en fines lors du processus de raffinement (cf. Sec. 3.4 p. 84). Pour cela, il nous sera nécessaire de disposer d'une notion de simulation adaptée à des machines dont les alphabets appartiennent à des abstractions de types de données différentes.

Soient les abstractions de types de données $\mathcal{T}_A, \mathcal{T}_B, \mathcal{T}_C, \mathcal{T}_D \in \mathfrak{D}^\alpha$ suivantes.

$\mathcal{T}_A$	=	$\langle In_A, \preccurlyeq_A, \sqcup_A, \sqcap_A, \top_A, \bot_A \rangle$	$\mathcal{T}_B$	=	$\langle In_B, \preccurlyeq_B, \sqcup_B, \sqcap_B, \top_B, \bot_B \rangle$	*(entrées)*
$\mathcal{T}_C$	=	$\langle Out_C, \preccurlyeq_C, \sqcup_C, \sqcap_C, \top_C, \bot_C \rangle$	$\mathcal{T}_D$	=	$\langle Out_D, \preccurlyeq_D, \sqcup_D, \sqcap_D, \top_D, \bot_D \rangle$	*(sorties)*

Dans la suite, nous considérons deux machines :

$$\begin{aligned} M_1 &= \langle In_A, Out_C, Q_1, \mathsf{out}_1, E_1, q_0^1 \rangle, \\ M_2 &= \langle In_B, Out_D, Q_2, \mathsf{out}_2, E_2, q_0^2 \rangle. \end{aligned}$$



(a) Machine concrète (b) Machine abstraite

FIGURE 3.3 – Abstraction de machines concrètes

Les alphabets d'entrée et de sortie de ces deux machines sont chacun reliés par une correspondance de Galois :

$$\begin{aligned} g_{In} &\equiv (\alpha_{In}, \gamma_{In}) \in Galois(\mathcal{T}_A, \mathcal{T}_B), \\ g_{Out} &\equiv (\alpha_{Out}, \gamma_{Out}) \in Galois(\mathcal{T}_C, \mathcal{T}_D). \end{aligned}$$

Cette définition est une transposition directe de celle donnée en Sec. 2.2.3. Cependant, une différence notable avec la relation de simulation "concrète" est que nous *devons* considérer ici des relations de simulation entre les espaces d'états de machines distinctes.

Définition 20 (Relation de simulation abstraite)**.** *Soit $R \subseteq Q_1 \times Q_2$ une relation sur les états de M_1 et M_2. Nous dirons que R est une* relation de simulation abstraite *si et seulement si, pour tout couple d'états $(q_1, q_2) \in R$:*

$$\Big(\forall q_1', q_1 \xrightarrow{a_1} q_1' \in E \implies \exists q_2', q_2 \xrightarrow{a_2} q_2' \in E \wedge a_1 \preccurlyeq_A \gamma_{In}(a_2) \ \wedge \ (q_1', q_2') \in R \Big) \ \wedge \\ \forall (q_1, q_2) \in R, \mathsf{out}_1(q_1) \preccurlyeq_C \gamma_{Out}(\mathsf{out}_2(q_2)).$$

Cette situation sera notée $q_1 \sqsubseteq_{g_{In}, g_{Out}, R} q_2$. Si il existe une telle relation R t.q. $(q_1, q_2) \in R$, nous dirons que q_2 simule q_1, ce que nous noterons $q_1 \sqsubseteq_{g_{In}, g_{Out}} q_2$. Lorsque le contexte le permettra, nous écrirons simplement $q_1 \sqsubseteq q_2$.

Exemple 3.10 : La figure 3.3 présente un exemple de machines en situation de simulation abstraite. La machine concrète procède au calcul du carré de chaque entrée. L'état observable initial est 0. La relation de simulation est $\{(q_0, p_0); (q_1, p_1)\}$. Nous prouvons ce fait par décomposition.

- Commençons par prouver que $q_1 \sqsubseteq p_1$. Soit la relation $R_1 = \{(q_1, p_1)\}$. Les transitions partant de q_1 sont dans l'ensemble $\{q_1 \xrightarrow{in} q_1 \in E \mid in \in \textbf{int}\}$. Vérifions que chacune de ces transitions est simulée par la transition $p_1 \xrightarrow{\top} p_1$:
  1. en entrée, nous avons de façon directe $\forall in \in In, in \in \gamma_S(\top)$;
  2. en sortie, nous avons $x^2 + 1 \in [1; max\_int]$, ce qui nous permet d'affirmer que $\alpha_S(x^2 + 1) = +$ et donc $x^2 + 1 \in \gamma_S(+)$;
  3. les états atteignables restent dans R_1. La relation R_1 est bien une relation de simulation.
- Soit la relation $R_0 = \{(q_0, p_0)\} \bigcup R_1$. Chaque transition de l'ensemble $\{q_0 \xrightarrow{in} q_1 \in E \mid in \in \textbf{int}\}$ est simulée par la transition $p_0 \xrightarrow{\top} p_1$, car $q_1 \sqsubseteq p_1$. De plus, $0 \in \gamma_S(\underline{0})$. La relation R_0 est bien une relation de simulation, et $q_0 \sqsubseteq p_0$.

Proposition 3.11. *Comme la relation de simulation sur les machines concrètes, la relation de simulation abstraite forme un pré-ordre.*

Démonstration. La réflexivité de $\sqsubseteq$ provient directement de la réflexivité des relations d'ordre $\preccurlyeq$ des abstractions de types de données. La transitivité de la simulation abstraite découle d'une part de la transitivité de la composition des correspondances de Galois (Def. 16) et d'autre part de la propriété correspondante de la simulation classique. □

Le pré-ordre de simulation abstraite sera le cadre de notre étude du délai de séparabilité relativement aux notions d'abstraction (cf. Sec. 3.2.4 p. 65) et de raffinement (Sec. 3.4 p. 84).

3.2.3.3 Fonction de concrétisation de machines sur treillis

Afin de lier les résultats de ce chapitre aux cas des machines concrètes (Chapitre 2), nous allons mettre en évidence l'existence d'une fonction du pré-ordre de simulation abstraite vers le pré-ordre de simulation concret préservant leur structure. Cette fonction prend la forme d'une fonction associant une machine concrète à toute machine sur treillis. Considérons les abstractions de types de données $\mathcal{T}_A, \mathcal{T}_B, \mathcal{T}_C, \mathcal{T}_D \in \mathfrak{D}^\alpha$ posées en Sec. 3.2.3.2 p. 61.

Supposons ces deux couples d'abstractions de types de données reliés entre eux par deux correspondances de Galois :

$$\begin{array}{rcl} g_{In} & \in & Galois(\mathcal{T}_A, \mathcal{T}_B), \\ g_{Out} & \in & Galois(\mathcal{T}_C, \mathcal{T}_D). \end{array}$$

Supposons enfin l'existence de types concrets $In, Out \in \mathfrak{D}$ tel que $\mathcal{T}_A$ et $\mathcal{T}_B$ soient en correspondance de Galois avec le treillis des parties de In, et de même pour les données en sortie :

$$\begin{array}{ll} h_A \in Galois(\mathcal{T}_{\wp(In)}, \mathcal{T}_A), & h_B \in Galois(\mathcal{T}_{\wp(In)}, \mathcal{T}_B), \\ h_C \in Galois(\mathcal{T}_{\wp(Out)}, \mathcal{T}_C), & h_D \in Galois(\mathcal{T}_{\wp(Out)}, \mathcal{T}_D). \end{array}$$

Cette situation est résumée en Fig. 3.4(a). Nous pouvons alors associer à toute machine de Moore sur treillis une machine de Moore sur des alphabets concrets, de sorte que la structure de pré-ordre soit conservée. En essence, cela nous permet d'affirmer que les développements que nous effectuons avec les machines sur treillis correspondent bien à des développements sur des machines de Moore simples, et cela nous autorise à nous servir des résultats sur la conservation du délai de séparbilité démontrés à la fin du chapitre précédent. La proposition suivante capture cette intuition.

Proposition 3.12 (Existence d'une fonction des machines abstraites vers les machines de Moore)**.** *Il existe une fonction depuis le pré-ordre de simulation abstraite vers le pré-ordre de simulation préservant la relation de pré-ordre :*

$$\begin{array}{l} \exists k_1 \in \mathsf{Moore}(\mathcal{T}_A, \mathcal{T}_C) \to \mathsf{Moore}(In, Out), \\ \exists k_2 \in \mathsf{Moore}(\mathcal{T}_B, \mathcal{T}_D) \to \mathsf{Moore}(In, Out), \\ \quad \forall M_1 \in \mathsf{Moore}(\mathcal{T}_A, \mathcal{T}_C), \forall M_2 \in \mathsf{Moore}(\mathcal{T}_B, \mathcal{T}_D), \\ \quad\quad M_1 \sqsubseteq_{g_{In}, g_{Out}} M_2 \implies k_1(M_1) \precsim k_2(M_2). \end{array}$$

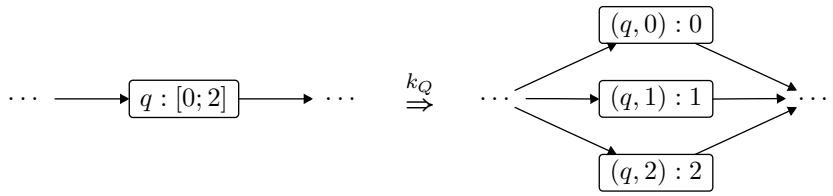
La situation est représentée en Fig. 3.4(b).

Nous effectuons la preuve dans le cas simplifié où $\mathcal{T}_A = \mathcal{T}_B$ et $\mathcal{T}_C = \mathcal{T}_D$. La preuve dans le cas général est similaire. Nous avons donc $g_{In} = (id, id)$, $g_{Out} = (id, id)$, $h_A = h_B = (\alpha_{In}, \gamma_{In})$ et $h_C = h_D = (\alpha_{Out}, \gamma_{Out})$. De plus, les fonctions k_1 et k_2 coincident : $k_1 = k_2 = k$.

Démonstration. Nous commençons par définir la fonction k du pré-ordre de simulation abstraite vers le pré-ordre concret. Soit $M = \langle In_A, Out_C, Q, \mathsf{out}, E, q_0\rangle$ une machine sur treillis. La fonction k associe à tout état abstrait un ensemble d'états concrets, et à toute transition un ensemble de transitions concrètes :

$$k(M) = \langle In, Out, \bigcup_{q\in Q} k_Q(q), \mathsf{out}_k, \bigcup_{(q,in,q')\in E} k_E(q, in, q'), Q_0\rangle.$$

Dans le cas des états, la fonction de concrétisation k_Q agit en créant une copie de l'état d'origine pour chaque valeur concrète correspondant au symbole de sortie, comme dans l'exemple qui suit.



L'image d'un arc $q \xrightarrow{in} q'$ par k_E est alors naturellement calculé comme l'ensemble d'arcs concrets entre $k_Q(q)$ et $k_Q(q')$. Les composantes de k sont définies comme suit :

$$\begin{array}{lcl} k_Q(q) & = & \{(q, output) \in Q \times Out \mid output \in \gamma_{Out}(\mathsf{out}(q))\}, \\ \mathsf{out}_k(q, output) & = & output, \\ k_E(q, in, q') & = & \left\{ (q_k, input, q'_k) \;\middle|\; \begin{array}{l} q_k \in k_Q(q), \\ q'_k \in k_Q(q'), \\ input \in \gamma_{In}(in) \end{array} \right\}, \\ Q_0 & = & k_Q(q_0). \end{array}$$

Il nous reste à prouver que k préserve la relation de simulation. Soient deux machines sur treillis :

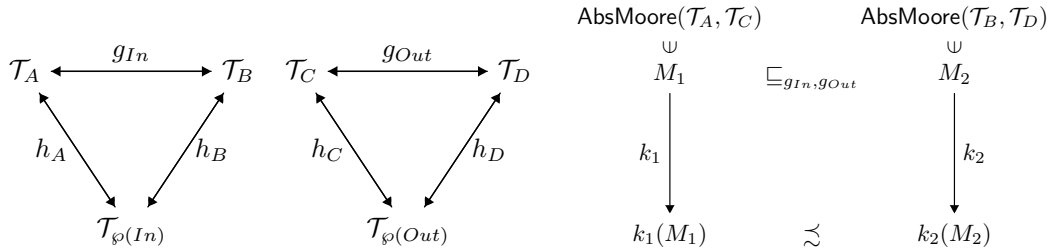
$$M_1 = \langle In_A, Out_C, Q_1, \mathsf{out}_1, E_1, q_{i,1}\rangle \quad M_2 = \langle In_A, Out_C, Q_2, \mathsf{out}_2, E_2, q_{i,2}\rangle$$

et soient deux états $q_1 \in Q_1, q_2 \in Q_2$ tels que $q_1 \sqsubseteq q_2$. Par définition, q_1 a pour image par k l'ensemble d'états $\{(q_1, output_1) \mid output_1 \in \gamma_{Out}(\mathsf{out}(q_1))\}$. Qui plus est, toute transition de la machine concrète $(q_1, output_1) \xrightarrow{input_1} (q'_1, output'_1)$ est obtenue comme image d'une transition $q_1 \xrightarrow{in_1} q'_1$ avec $input_1 \in \gamma_{In}(in_1)$. Pour toute telle transition $q_1 \xrightarrow{in_1} q'_1$, il existe par simulation abstraite une transition $q_2 \xrightarrow{in_2} q'_2$. Par application de la fonction de concrétisation, il existe dans les transitions concrètes obtenues par k depuis $q_2 \xrightarrow{in_2} q'_2$ un ensemble de transitions :

$$\{((q_2, output_2), input_1, (q'_2, output'_2))\}$$

permettant de conclure. □

Exemple 3.13 (Application de la fonction de concrétisation) **:** La figure 3.2.3.3 montre un exemple d'application de la fonction de concrétisation. La machine sur treillis, constituée de deux états q_0, q_1, calcule à chaque instant une fonction conservant le signe de l'entrée. La fonction de concrétisation associe à q_0 un ensemble d'états $\{(q_0, i) \mid i \in [min\_int,\ -1]\}$ totalement connectés par tous les arcs de l'ensemble $k_E(q_0, [min\_int; -1], q_0)$, et similairement pour q_1. Les ensembles d'arcs $k_E(q_0, [0; max\_int], q_1)$ et $k_E(q_1, [min\_int; -1], q_0)$ sont représentés synthétiquement, pour des raisons de lisibilité.



(a) Correspondances de Galois abstraites et concrètes

(b) Fonction du pré-ordre de simulation abstraite vers le pré-ordre concret

FIGURE 3.4 – Fonction entre les pré-ordres de simulation abstraite et de simulation concrète

3.2.4 Délai de séparabilité sur machines sur treillis

Nous devons donner une notion de délai de séparabilité qui soit adaptée aux machines sur treillis décrites plus haut. Les définitions qui suivent sont conçues pour permettre la preuve de la conservation de propriétés des machines sur treillis vers les machines concrètes par l'opération de concrétisation. De la sorte, nous voulons montrer que le cadre des machines sur treillis est adapté à l'analyse du délai de séparabilité. Dans la suite, nous considérons deux abstractions de types de données :

$$\begin{array}{lcll} \mathcal{T}_I & = & \langle I, \preccurlyeq_I, \sqcup_I, \sqcap_I, \top_I, \bot_I \rangle & \textit{(entrées)} \\ \mathcal{T}_O & = & \langle O, \preccurlyeq_o, \sqcup_o, \sqcap_o, \top_o, \bot_o \rangle & \textit{(sorties)} \end{array}$$

ainsi qu'une machine sur ces treillis $M = \langle I, O, Q, \mathsf{out}, E, q_0 \rangle$. Observons également que puisque $\mathcal{T}_I$ et $\mathcal{T}_O$ sont des abstractions de types de données, il doit exister deux types de données concrets $In, Out \in \mathfrak{D}$ tels qu'il existe deux correspondances de Galois :

$$\begin{array}{lcl} (\alpha_{In}, \gamma_{In}) & \in & Galois(\mathcal{T}_{\wp(In)}, \mathcal{T}_I), \\ (\alpha_{Out}, \gamma_{Out}) & \in & Galois(\mathcal{T}_{\wp(Out)}, \mathcal{T}_O). \end{array}$$

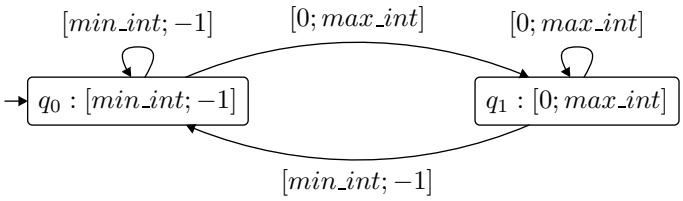
3.2.4.1 Séparabilité abstraite

Dans cette section, nous proposons des notions de séparabilité adaptées au cadre des machines sur treillis, et nous montrons que ces notions impliquent les propriétés correspondantes sur les machines concrètes. Nous commençons par définir une sous-approximation abstraite des contre-exemples à la bisimilarité, analogue à celle proposée en Sec. 2.3.2.

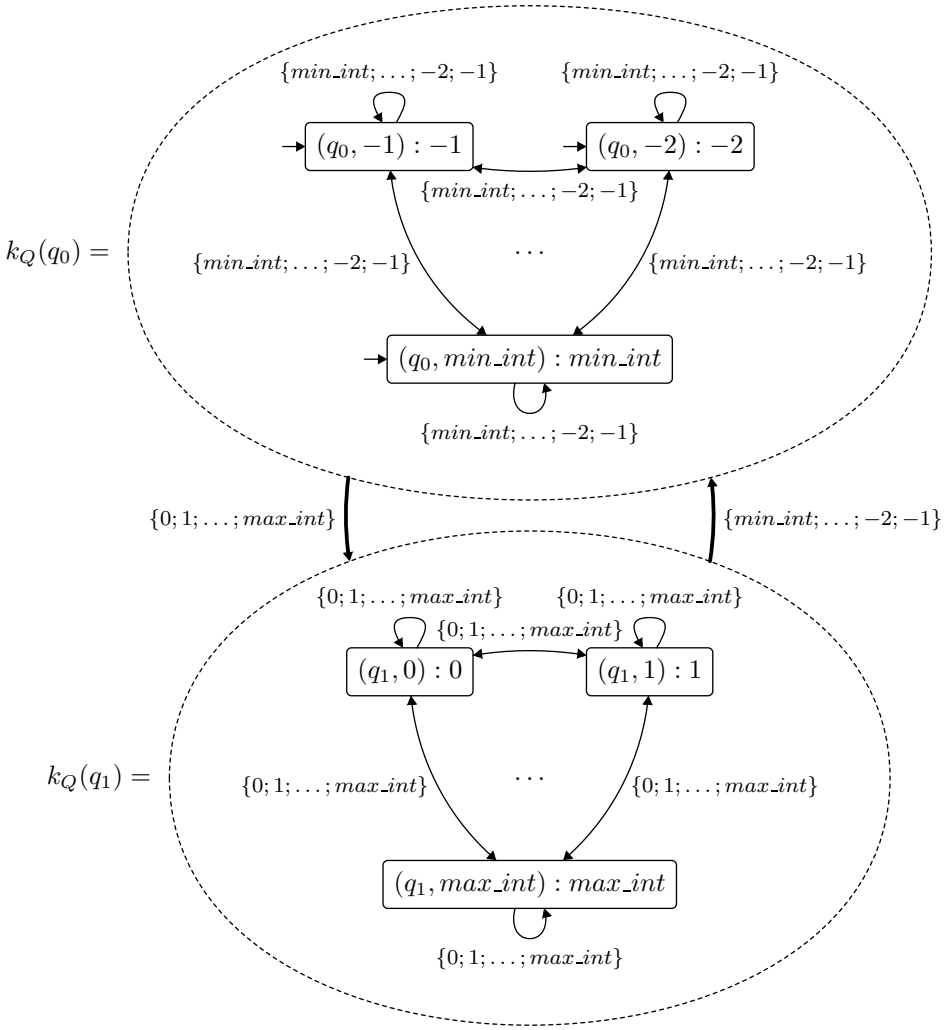
Proposition 3.14 (Sous-approximation de la non-bisimilarité)**.** *Nous définissons inductivement un ensemble* $\mathsf{c{-}ex}^\alpha \subseteq Q \times Q$ *de contre-exemples à la bisimilarité de deux états. Les règles sont données ci-dessous, où* p *et* q *sont universellement quantifiées.*

$$\text{ABS-BASE}\ \frac{}{\mathsf{out}(p) \sqcap_o \mathsf{out}(q) = \bot_o \implies \mathsf{c{-}ex}^\alpha(p,q)} \qquad \text{ABS-SYM}\ \frac{\mathsf{c{-}ex}^\alpha(q,p)}{\mathsf{c{-}ex}^\alpha(p,q)}$$

$$\text{ABS-IND}\ \frac{\exists a_p \in I, \exists p' \in Q, \exists p \xrightarrow{a_p} p' \in E, \forall q' \in Q, \forall a_q \preccurlyeq_I a_p, q \xrightarrow{a_q} q' \in E \implies \mathsf{c{-}ex}^\alpha(p',q')}{\mathsf{c{-}ex}^\alpha(p',q')}$$



(a) Abstraction de machine de Moore



(b) Concrétisation de machine de Moore

FIGURE 3.5 – Concrétisation d'une abstraction de machine de Moore

La propriété suivante est vérifiée par $\mathsf{c{-}ex}^{\alpha}$ *:*

$$\mathsf{c{-}ex}^{\alpha}(p,q) \implies \forall p_k \in k_Q(p), \forall q_k \in k_Q(q), p_k \not\sim q_k$$

Cette propriété est prouvée en construisant à partir de tout contre-exemple abstrait de $\mathsf{c{-}ex}^{\alpha}(p,q)$ un ensemble de contres-exemples concrets.

Démonstration. Nous procédons par induction sur $\mathsf{c{-}ex}^{\alpha}(p,q)$.

- (Cas ABS-BASE.) Nous avons $\mathsf{out}(p) \sqcap_o \mathsf{out}(q) = \bot_o$. Soient $p_k \in k_Q(p)$ et $q_k \in k_Q(q)$ deux états concrets. Nous allons prouver que $\mathsf{out}(p_k) \neq \mathsf{out}(q_k)$ en montrant :

  $$\gamma_{Out}(\mathsf{out}(p)) \bigcap \gamma_{Out}(\mathsf{out}(q)) = \varnothing.$$

  - Nous commençons par montrer la proposition suivante :

    $$\forall a, b, \gamma_{Out}(a \sqcap_o b) = \gamma_{Out}(a) \bigcap \gamma_{Out}(b).$$

    Par monotonicité de γ_{Out}, nous obtenons :

    $$\gamma_{Out}(a \sqcap_o b) \subseteq \gamma_{Out}(a) \ \wedge \ \gamma_{Out}(a \sqcap b) \subseteq \gamma_{Out}(b).$$

    Supposons qu'il existe un c tel que $c \subseteq \gamma_{Out}(a \sqcap b)$. Par transitivité, nous avons :

    $$c \subseteq \gamma_{Out}(a) \wedge c \subseteq \gamma_{Out}(b).$$

    Par propriété des correspondances de Galois, ceci entraîne :

    $$\alpha_{Out}(c) \preccurlyeq_o a \ \wedge \ \alpha_{Out}(c) \preccurlyeq_o b.$$

    Par définition de $\sqcap_o$, $a \sqcap_o b \preccurlyeq \alpha_{Out}(c)$. Nous obtenons par une nouvelle application de la propriété des correspondances de Galois $\gamma_{Out}(a \sqcap_o b) \subseteq c$, ce qui montre que $\gamma_{Out}(a \sqcap_o b)$ est la borne inférieure de a et b, c'est-à-dire :

    $$\gamma_{Out}(a \sqcap_o b) = \gamma_{Out}(a) \bigcap \gamma_{Out}(b).$$

  - La propriété $\gamma_{Out}(\bot_o) = \varnothing$ est directement obtenue de la définition des correspondances. Nous avons au final :

    $$\gamma_{Out}(\mathsf{out}(p) \sqcap_o \mathsf{out}(q)) = \gamma_{Out}(\mathsf{out}(p)) \bigcap \gamma_{Out}(\mathsf{out}(q)) = \gamma_{Out}(\bot_o) = \varnothing.$$

    Ceci conclut la preuve pour le cas ABS-BASE.

- (Cas ABS-IND.) Nous avons comme hypothèse :

  $$\exists a_p \in I, \exists p' \in Q, \exists p \xrightarrow{a_p} p' \in E, \forall q' \in Q, \forall a_q \preccurlyeq a_p, q \xrightarrow{a_q} q' \in E \implies \mathsf{c{-}ex}^{\alpha}(p', q').$$

  Soient $k_Q(p)$, $k_Q(p')$ et $k_Q(q)$ les ensembles d'états concrets respectivement associés à p, p' et q. L'arc $p \xrightarrow{a_p} p'$ correspond à l'ensemble d'arcs concrets suivants :

  $$k_E(p \xrightarrow{a_p} p') = \{(p_k, input, p'_k) \mid p_k \in k_Q(p) \ \wedge \ p'_k \in k_Q(p') \ \wedge \ input \in \gamma_{In}(a_p)\}.$$

  De même, chaque arc $q \xrightarrow{a_q} q'$ tel que $a_q \preccurlyeq_I a_p$ correspond à l'ensemble d'arcs concrets suivants :

  $$k_E(q \xrightarrow{a_q} q') = \{(q_k, input, q'_k) \mid q_k \in k_Q(q) \ \wedge \ q'_k \in k_Q(q') \ \wedge \ input \in \gamma_{In}(a_q)\}.$$

  Par hypothèse d'induction, nous avons dans tous les cas $p'_k \not\sim q'_k$. Pour tout arc $p_k \xrightarrow{input} p'_k$ et tout arc $q_k \xrightarrow{input} q'_k$, nous pouvons donc construire par la règle IND (Def. 2.7 p. 35) une preuve de $p_k \not\sim q_k$. Le cas de la règle ABS-SYM est direct.

□

A partir de la sous-approximation de la non-bisimilarité définie plus haut, nous pouvons proposer une notion de paire séparante abstraite, définie de façon semblable à sa contrepartie concrète. Afin de garantir que toute paire séparante abstraite (a_1, a_2) corresponde à un ensemble de paires séparantes concrètes, nous imposons que les symboles a_1 et a_2 soient disjoints.

Définition 21 (Paires séparantes abstraites)**.** *Soit $q \in Q$ un état. L'ensemble des paires séparantes abstraites de q est noté* $\mathsf{SepPairs}^\alpha(q)$ *et est défini comme suit :*

$$\mathsf{SepPairs}^\alpha(q) = \left\{ (a_1,a_2) \in I \;\middle|\; \begin{array}{l} a_1 \sqcap_I a_2 = \bot_I \;\wedge \\ \exists q_1 \in Q, \exists q \xrightarrow{a_1} q_1 \in E, \\ \quad \forall b_2 \preccurlyeq_I a_2, \forall q_2 \in Q, q \xrightarrow{b_2} q_2 \in E \implies \mathsf{c\text{-}ex}^\alpha(q_1,q_2) \vee \\ \exists q_1 \in Q, \exists q \xrightarrow{a_2} q_1 \in E, \\ \quad \forall b_1 \preccurlyeq_I a_1, \forall q_2 \in Q, q \xrightarrow{b_1} q_2 \in E \implies \mathsf{c\text{-}ex}^\alpha(q_1,q_2) \end{array} \right\}$$

La preuve qu'une paire séparante abstraite induit par concrétisation un ensemble de paires séparantes concrètes est un corollaire direct de la proposition 3.14.

La définition de séparabilité abstraite donne lieu à la notion usuelle de séquence d'exécution séparante, et les différents types de séparateurs qui y sont liés. La méthode de définition de ces séquences d'exécution séparantes est en tous points semblable au cas concret (Def. 7 p. 36), nous en omettons donc les détails.

Définition 22 (Séquences d'exécution séparantes et séparateurs abstraits)**.** *Soient $p, q \in Q$ deux états et soit $\mathsf{c\text{-}ex}^\alpha(p,q)$ l'ensemble des contre-exemples abstrait à la bisimilarité de p et q. Nous supposons cet ensemble non-vide. A tout contre-exemple $h \in \mathsf{c\text{-}ex}^\alpha(p,q)$, nous pouvons associer l'ensemble des séquences d'exécution séparantes induites par h, défini comme l'ensemble des chemins sur l'arbre de dérivation h, et noté $\mathsf{SepEx}^\alpha(h)$. L'ensemble des séquences d'exécution séparantes de p et q est alors définie comme suit :*

$$\mathsf{SepEx}^\alpha(p,q) = \bigcup_{h \in \mathsf{c\text{-}ex}^\alpha(p,q)} \mathsf{SepEx}^\alpha(h).$$

Chaque séquence d'exécution séparante abstraite induit un mot appelé séparateur abstrait*. L'ensemble des séparateurs abstraits de p et q est noté $\mathsf{S}^\alpha(p,q)$. Comme dans le cas concret, nous pouvons distinguer les séparateurs abstraits effectifs $\mathsf{S}^\alpha_!(p,q)$ et les pseudo-séparateurs abstraits $\mathsf{S}^\alpha_*(p,q)$.*

$$\begin{array}{lcl} \mathsf{S}^\alpha(p,q) & = & \{w \mid (r_1,r_2) \in \mathsf{SepEx}^\alpha(p,q) \;\wedge\; r_1 \in \mathsf{Ex}^\alpha(p,w), r_2 \in \mathsf{Ex}^\alpha(q,w)\} \\ \mathsf{S}^\alpha_!(p,q) & = & \{w \mid \forall r_1 \in \mathsf{Ex}^\alpha(p,w), \forall r_2 \in \mathsf{Ex}^\alpha(q,w), (r_1,r_2) \in \mathsf{SepEx}^\alpha(p,q)\} \\ \mathsf{S}^\alpha_*(p,q) & = & \mathsf{Minimal}\{w \mid \mathcal{L}^*(p,w) \overline{\cap} \mathcal{L}^*(q,w) = \varnothing\} \end{array}$$

Corollaire 3.15 (Conservation des séparateurs)**.** *Une conséquence directe de la proposition 3.14 est que pour tous états abstraits p, q et séparateur abstrait $w \in \mathsf{S}^\alpha(p,q)$, si $\gamma^*(w)$ est l'ensemble de mots concrets correspondant à w, alors :*

$$\forall p_k \in k_Q(p), q_k \in k_Q(q), \gamma^*(w) \subseteq \mathsf{S}(p_k, q_k).$$

En d'autres termes, les séparateurs abstraits sous-approximent les séparateurs concrets. L'inclusion est également vérifiée dans le cas des séparateurs effectifs :

$$\forall p_k \in k_Q(p), q_k \in k_Q(q), \gamma^*(\mathsf{S}^\alpha_!(p,q)) \subseteq \mathsf{S}(p_k, q_k).$$

L'inclusion n'est pas vérifiée dans le cas des pseudo-séparateurs, à cause de leur condition de minimalité.

3.2.4.2 Délai de séparabilité abstrait

Les séparateurs abstraits nous permettent à nouveau de transposer la notion de délai de séparabilité dans le cadre des machines sur treillis. Les définitions des délais de séparabilité optimiste et pessimiste sur les machines abstraites sont semblables aux définitions du cas concret. Commençons par étudier le délai de séparabilité abstrait dans le cas optimiste.

Délai de séparabilité optimiste abstrait. Soient $q_1, q_2 \in Q$ deux états tels que $\mathsf{c{-}ex}^\alpha(q_1, q_2)$. Le délai de séparabilité optimiste abstrait de q_1 et q_2, noté $\mathsf{septime}^\alpha_{\mathsf{O}}(q_1, q_2)$, est défini comme la longueur du plus petit séparateur abstrait de q_1 et q_2 :

$$\mathsf{septime}^\alpha_{\mathsf{O}}(q_1, q_2) = \mathsf{min}\ \{|w| \mid w \in \mathsf{S}^\alpha(q_1, q_2)\}.$$

Le délai de séparabilité d'un état $q \in Q$ est alors défini comme dans le cas concret, en calculant le minimum du délai sur l'ensemble des paires séparantes abstraites de q :

$$\mathsf{septime}^\alpha_{\mathsf{O}}(q) = \mathsf{min} \left\{ \mathsf{septime}^\alpha_{\mathsf{O}}(q_1, q_2) \;\middle|\; \begin{array}{l} \exists (a_1, a_2) \in \mathsf{SepPairs}^\alpha(q), \\ \exists b_1 \preccurlyeq_I a_1, b_2 \preccurlyeq_I a_2, \\ q \xrightarrow{b_1} q_1 \in E \ \wedge\ q \xrightarrow{b_2} q_2 \in E \end{array} \right\}.$$

Nous pouvons montrer que le délai de séparabilité optimiste abstrait sur-approxime le délai de séparabilité optimiste concret.

Proposition 3.16 (Sur-approximation du délai de séparabilité optimiste concret)**.** *Ce délai sur-approxime sa contrepartie concrète :*

$$\forall q_k \in k_Q(q), \mathsf{septime}_{\mathsf{O}}(q_k) \leq \mathsf{septime}^\alpha_{\mathsf{O}}(q).$$

Démonstration. Nous allons commencer par prouver que les paires séparantes abstraites sous-approximent les états atteignables par les paires séparantes des états concrets. Afin de prouver que le délai de séparabilité abstrait majore le délai concret, il faut et il suffit de montrer que le délai de séparabilité de toute paire séparante abstraite est minoré par le délai de séparabilité d'une paire séparante concrète.

Soit $(a_1, a_2) \in \mathsf{SepPairs}^\alpha(q)$ une paire séparante quelconque. L'ensemble des paires concrètes y étant associé est :

$$\gamma_{In}(a_1, a_2) = \{(x_1, x_2) \mid x_1 \in \gamma_{In}(a_1), x_2 \in \gamma_{In}(a_2)\}.$$

La propriété à prouver peut s'écrire :

$$\exists (x_1, x_2) \in \gamma_{In}(a_1, a_2), \mathsf{septime}_{\mathsf{O}}(x_1, x_2) \leq \mathsf{septime}^\alpha_{\mathsf{O}}(a_1, a_2).$$

Nous choisissons un état $q_k \in Q_k(q)$, et une paire $(x_1, x_2) \in \gamma_{In}(a_1, a_2)$ *arbitrairement*. Soient

$$Q_1 = \{q_{k,1} \mid q_k \xrightarrow{x_1} q_{k,1} \in k_E(E)\} \quad Q_2 = \{q_{k,2} \mid q_k \xrightarrow{x_2} q_{k,2} \in k_E(E)\}$$

les états concrets atteignables respectivement par x_1 et x_2. Nous devons prouver :

$$\min \left\{ \mathsf{septime}_{\mathsf{O}}(q_{k,1}, q_{k,2}) \;\middle|\; \begin{array}{l} q_{k,1} \in Q_1 \\ q_{k,2} \in Q_2 \end{array} \right\} \leq \min \left\{ \mathsf{septime}^{\alpha}_{\mathsf{O}}(q_1, q_2) \;\middle|\; \begin{array}{l} \exists b_1 \preccurlyeq_I a_1, b_2 \preccurlyeq_I a_2, \\ q \xrightarrow{b_1} q_1 \in E \wedge \\ q \xrightarrow{b_2} q_2 \end{array} \right\}.$$

Nous pouvons à nouveau procéder en montrant que tout élément $\mathsf{septime}^{\alpha}_{\mathsf{O}}(q_1, q_2)$ est minoré par un élément $\mathsf{septime}_{\mathsf{O}}(q_{k,1}, q_{k,2})$. Par définition de la concrétisation de machines sur treillis, toute transition $q_k \xrightarrow{x_i} q_{k,i}$ appartient à l'image d'une transition $q \xrightarrow{c_i} q_i$ ($c_i \preccurlyeq a_i \wedge i \in \{1; 2\}$). Or, par le corollaire 3.15, tout séparateur abstrait de deux états $q_1, q_2 \in Q$ induit un ensemble de séparateurs concrets de même longueur pour tous les couples $(q_{k,1}, q_{k,2})$. De ce fait, nous avons dans le pire des cas l'égalité $\mathsf{septime}_{\mathsf{O}}(q_{k,1}, q_{k,2}) = \mathsf{septime}^{\alpha}_{\mathsf{O}}(q_1, q_2)$, ce qui clos la preuve. □

Délai de séparabilité pessimiste abstrait. Soient p, q deux états tels que $\mathsf{c{-}ex}^{\alpha}(p, q)$. La définition du délai de séparabilité pessimiste abstrait de p et q est à nouveau une adaptation du cas concret (Def. 10 p. 39). Dans le cas concret, nous avons défini les conditions nécessaires et suffisantes à ce que le délai de séparabilité pessimiste soit fini (Sec. 2.3.4.3 p. 41). Ces conditions inspirent une sur-approximation du délai de séparabilité pessimiste abstrait. Ce dernier est fini si tous les mots d'entrée infinis induisent des langages abstraits disjoints. Nous notons cette condition $p \;)\!(^{\alpha}\; q$:

$$p \;)\!(^{\alpha}\; q \iff \forall w \in I^{\omega}, \exists w' \in prefix(w), w' \in \mathsf{S}^{\alpha}_{*}(p, q).$$

Cette condition est préservée par concrétisation, comme le montre le lemme suivant.

Lemme 3.17. *Soient $p, q \in Q$ deux états abstraits. La propriété suivante est vérifiée :*

$$p \;)\!(^{\alpha}\; q \iff \forall p_k \in k_Q(p), \forall q_k \in k_Q(q), p_k \;)\!(\; q_k.$$

Démonstration.

- Supposons $p \;)\!(^{\alpha}\; q$. Soient $p_k \in k_Q(p)$ et $q_k \in k_Q(p)$ deux états concrets. Nous raisonnons par l'absurde afin de prouver $p_k \;)\!(\; q_k$. Supposons qu'il existe un mot d'entrée w infini ne générant pas de langages disjoints pour p_k et q_k. Il existe donc au moins deux séquences d'exécution infinies :

  $$p_k = p_{k,0} \xrightarrow{w[0]} p_{k,1} \xrightarrow{w[1]} \dots \quad q_k = q_{k,0} \xrightarrow{w[0]} q_{k,1} \xrightarrow{w[1]} \dots$$

  telles que $\forall i, \mathsf{out}(p_{k,i}) = \mathsf{out}(q_{k,i})$. Par définition de la fonction de concrétisation k_E, il doit exister deux séquences correspondantes dans la machine abstraite :

  $$p = p_0 \xrightarrow{a_0} p_1 \xrightarrow{a_1} \dots \quad q = q_0 \xrightarrow{b_0} q_1 \xrightarrow{b_1} \dots$$

  telles que $\forall i, w[i] \in a_i \sqcap_I b_i \wedge (\mathsf{out}(p_i) \sqcap_o \mathsf{out}(q_i) \neq \bot_o)$. Ceci est en contradiction avec l'hypothèse $p \;)\!(^{\alpha}\; q$, ce qui prouve l'implication.

- Soient $p_k \in k_Q(p), q_k \in k_Q(q)$ deux états concrets tels que $p_k \not)\!(\ q_k$. Nous prouvons $p \not)\!(^\alpha q$ par l'absurde. Si $\neg(p \not)\!(^\alpha q)$, alors il existe un mot w et au moins deux séquences d'exécution abstraites infinies :

$$p = p_0 \xrightarrow{w[0]} p_1 \xrightarrow{w[1]} p_2 \dots$$
$$q = q_0 \xrightarrow{w[0]} q_1 \xrightarrow{w[1]} q_2 \dots$$

telles que $\forall i, \mathsf{out}(p_i) \sqcap_o \mathsf{out}(q_i) \neq \perp_o$. En utilisant cette propriété en conjonction avec la définition de la fonction de concrétisation, nous pouvons construire deux séquences d'exécutions concrètes :

$$p_k = p_{k,0} \xrightarrow{w[0]} p_{k,1} \xrightarrow{w[1]} p_{k,2} \dots$$
$$q_k = q_{k,0} \xrightarrow{w[0]} q_{k,1} \xrightarrow{w[1]} q_{k,2} \dots$$

telles que $\forall i, p_{k,i} \in k_Q(p_i)$ et $\forall i > 0, \mathsf{out}(p_{k,i}) = \mathsf{out}(q_{k,i})$. Ceci est en contradiction avec l'hypothèse $p_k \not)\!(\ q_k$, ce qui achève de prouver le lemme.

□

Nous notons le délai de séparabilité pessimiste abstrait de deux états p et q par $\mathsf{septime}^\alpha_{\mathsf{P}}(p, q)$, et le définissons par analyse par cas.

$$\begin{array}{lcl} \mathsf{septime}^\alpha_{\mathsf{P}}(p,q) & = & \infty \text{ si } \neg(p \not)\!(^\alpha q) \\ \mathsf{septime}^\alpha_{\mathsf{P}}(p,q) & = & \mathsf{max}\ \{|w| \mid \forall w' \in \mathsf{prefix}(w), w' \notin \mathsf{S}^\alpha(p,q)\} \text{ si } p \not)\!(^\alpha q. \end{array}$$

Nous pouvons alors prouver un résultat de sur-approximation du délai de séparabilité pessimiste, comme pour le cas optimiste. La preuve repose sur le fait que les pseudo-séparateurs abstraits induisent des pseudo-séparateurs concrets plus courts.

Proposition 3.18 (Sur-approximation du délai de séparabilité pessimiste concret)**.** *Soient deux états $p, q \in Q$. Le délai de séparabilité pessimiste de* deux états *calculé après abstraction sur-approxime le délai concret :*

$$\forall p_k \in k_Q(p), q_k \in k_Q(q), \mathsf{septime}_{\mathsf{P}}(p_k, q_k) \leq \mathsf{septime}^\alpha_{\mathsf{O}}(p, q).$$

Démonstration. Si $\neg(p \not)\!(^\alpha q)$, $\mathsf{septime}^\alpha_{\mathsf{P}}(p, q) = \infty$ et la sur-approximation est triviale. Sinon, tous les mots d'entrée sont des pseudo-séparateurs abstraits. Soient $p_k \in k_Q(p)$ et $q_k \in k_Q(q)$ deux états concrets quelconques. Afin de prouver la proposition, il suffit de montrer que l'existence de tout pseudo-séparateur abstrait implique l'existence d'un pseudo-séparateur concret de p_k et q_k de longueur inférieure ou égale. Soit un mot $w \in I^*$ induisant des langages abstraits disjoints. Il est direct de montrer que toute concrétisation $v \in \gamma^*_{In}(w)$ est telle que :

$$\mathcal{L}^*(p_k, v) \bigcap \mathcal{L}^*(q_k, v) = \varnothing.$$

Il existe donc un préfixe de v de taille minimale tel que $v \in \mathsf{S}_*(p_k, q_k)$ de taille nécessairement inférieure à w, ce qui achève de prouver la proposition. □

Afin d'étendre la sur-approximation du délai de séparabilité aux états, il resterait à montrer que le délai de séparabilité pessimiste de toute paire séparante concrète est sur-approximée par une paire séparante abstraite. Ce n'est pas le cas : étant donné un état $q \in Q$, une paire d'entrées non-séparante peut donner lieu par concrétisation à des paires séparantes concrètes dont le délai de séparabilité pessimiste est supérieur à celui calculé sur la machine sur treillis. La preuve de conservation ne s'étend donc pas aux états.

Nous pouvons résumer les résultats de cette section comme suit :

– il est possible de prouver le délai de séparabilité optimiste d'un ensemble de machines concrètes en calculant une abstraction de ces machines ;
– le cas pessimiste semble requérir un compromis avec les définitions, par exemple en acceptant de ne considérer qu'un sous-ensemble particulier des paires séparantes.

3.3 Réduction du délai de séparabilité contextuel au cas simple par abstraction

Toutes les notions présentées jusqu'ici considèrent les entrées-sorties de manière globale. Cela revient à supposer que ces systèmes ont un seul port d'entrée et un seul port de sortie. Se limiter à un tel mécanisme est insuffisant en pratique : il est courant pour un système temps-réel que les données transitant par des entrées et sorties distinctes aient des contraintes temporelles différentes. Par exemple, la prise en compte de certaines données peut être optionnelle. Spécifier une contrainte de délai de séparabilité dans ce cas sur-contraindrait le système. Il est utile de pouvoir raisonner à une granularité plus fine, en considérant seulement certaines composantes choisies sur les données en entrée comme en sortie. Cela pose immédiatement la question du statut des autres composantes, c'est-à-dire du *contexte*. Enfin, il n'est pas envisageable d'un point de vue calculatoire d'énumérer tous ces contextes pour calculer le délai de séparabilité. Il nous faut rechercher une solution à cet obstacle.

Cette section est dédiée dans un premier temps à la définition des notions de ports, de contexte et de délai de séparabilité contextuel (Sec. 3.3.1 et 3.3.2), préliminaires à l'étude d'une méthode d'abstraction procédant par effacement (Sec. 3.3.1.2 à 3.3.4) et permettant de prouver certains résultat d'approximation du délai de séparabilité (Sec. 3.3.5).

3.3.1 Ports d'entrée-sortie et notion de contexte

Nous avons défini les éléments contenus dans les flots d'entrée et de sortie de nos machines comme des n-uplets. Une notion naturelle de port est de sélectionner certaines composantes particulières de ces n-uplets. Cela revient à encoder un port par une fonction de projection sur les données, où les données sélectionnées par la projection sont par définition les données sur le *port* considéré.

3.3.1.1 Définition de la notion de port comme fonction de projection

Dans la définition des types de données pour les machines concrètes (Sec. 2.1.2 p. 24), il est fait référence à l'existence d'isomorphismes correspondants à l'associativité, la commutativité et l'absorption de l'élément neutre. Les ensembles des ports d'entrée et de sortie sont alors toutes les fonctions de projection possibles sur les données et les machines, modulo ces isomorphismes. Ceci nous permet de réduire notre étude au cas exposé ci-après.

Définition 23 (Ports d'entrée-sortie)**.** *Soit $A \times B$ le produit cartésien de deux ensembles A et B. Les deux ports sur $A \times B$ sont les deux projections $\pi_A(a, b) = a$ et $\pi_B(a, b) = b$.*

Il est possible d'étendre cette définition à des machines (concrètes ou sur treillis) à l'aide de deux fonctions de projections correspondant respectivement à un port d'entrée et un port de sortie.

Définition 24 (Projections de données)**.** *Soient In, Out, X, Y des ensembles finis quelconques et soit $M = \langle In \times X, Out \times Y, Q, E, \mathsf{out}, Q_i\rangle$ une machine. Nous définissons l'effacement des composantes X et Y dans M, dont la signature est :*

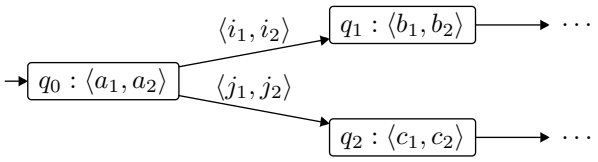
$$\prod[In, Out] : \mathsf{Moore}(In \times X, Out \times Y) \to \mathsf{Moore}(In, Out),$$

par l'équation suivante :

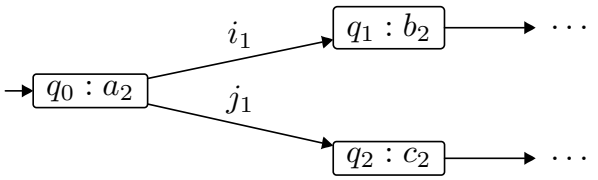
$$\prod[In, Out](M) = \langle In, Out, Q, \prod[In, Out](E'), \prod[In, Out](\mathsf{out}), Q_i\rangle ; \textit{où :}$$

- $\prod[In, Out](E) = \{(p, \pi_{In}(in), q) \mid (p, in, q) \in E\}$ *et*
- $\prod[In, Out](\mathsf{out}(q)) = \pi_{Out}(\mathsf{out}(q))$.

Exemple 3.19 : La figure 3.6 montre un exemple d'application d'une fonction de projection. Soit $M \in \mathsf{Moore}(In_1 \times In_2, Out_1 \times Out_2)$ la machine montrée en Fig. 3.6(a). Nous avons donc $i_1, j_1 \in In_1$, $i_2, j_2 \in In_2$, $a_1, b_1, c_1 \in Out_1$ et $a_2, b_2, c_2 \in Out_2$. Nous calculons la projection de M sur la première composante en entrée et sur la seconde composante en sortie, nommément $\prod[In_1, Out_2](M)$. Le résultat est montré en Fig. 3.6(b).



(a) Machine avant projection



(b) Machine après projection

FIGURE 3.6 – Exemple de projection sur une machine

Notons que la projection de donnée peut également être définie sur les dépliages (définis au chapitre 2, Sec. 2.2.4 p. 29). Considérons $\mathcal{T}_{In}, \mathcal{T}_{Out} \in \mathfrak{D}^{\alpha}$ deux abstractions de types de données :

$$\begin{array}{lcl}
\mathcal{T}_{In} & = & \langle In, \preccurlyeq_I, \sqcup_I, \sqcap_I, \top_I, \bot_I\rangle \\
\mathcal{T}_{Out} & = & \langle Out, \preccurlyeq_o, \sqcup_o, \sqcap_o, \top_o, \bot_o\rangle \\
\mathcal{T}^{+,-}_{In} & = & \langle In^{+,-}, \preccurlyeq^{+,-}_I, \sqcup^{+,-}_I, \sqcap^{+,-}_I, \top^{+,-}_I, \bot^{+,-}_I\rangle \\
\mathcal{T}^{+,-}_{Out} & = & \langle Out^{+,-}, \preccurlyeq^{+,-}_o, \sqcup^{+,-}_o, \sqcap^{+,-}_o, \top^{+,-}_o, \bot^{+,-}_o\rangle
\end{array}$$

Supposons que les équivalences $\mathcal{T}_{In} \simeq \mathcal{T}^{-}_{In} \times \mathcal{T}^{+}_{In}$ et $\mathcal{T}_{Out} \simeq \mathcal{T}^{-}_{Out} \times \mathcal{T}^{+}_{Out}$ soient vérifiées. Cela revient à partitionner chaque donnée en entrée en deux composantes, que nous désignerons "actives" et "passives".

Définition 25 (Ports actifs et passifs)**.** *Soient une paire de ports d'entrée $(\mathcal{A}_{In}, \mathcal{A}_{Out})$ et une paire de ports de sortie $(\mathcal{P}_{In}, \mathcal{P}_{Out})$:*

$$\begin{array}{ll|l}
\mathcal{A}_{In} : \mathcal{T}_{In} \to \mathcal{T}^{+}_{In} & \mathcal{P}_{In} : \mathcal{T}_{In} \to \mathcal{T}^{-}_{In} & \textit{(ports d'entrée)} \\
\mathcal{A}_{Out} : \mathcal{T}_{Out} \to \mathcal{T}^{+}_{Out} & \mathcal{P}_{Out} : \mathcal{T}_{Out} \to \mathcal{T}^{-}_{Out} & \textit{(ports de sortie)}
\end{array}$$

Supposons que ces ports forment une partition de $\mathcal{T}_{In}$ et $\mathcal{T}_{Out}$:

$$\lambda x.\langle \mathcal{A}_{In}(x), \mathcal{P}_{In}(x)\rangle \simeq id, \quad \lambda x.\langle \mathcal{A}_{Out}(x), \mathcal{P}_{Out}(x)\rangle \simeq id.$$

Les ports $\mathcal{A}_{In}$ et $\mathcal{A}_{Out}$ sont appelés ports actifs*, respectivement en entrée et en sortie. Les ports $\mathcal{P}_{In}$ et $\mathcal{P}_{Out}$, complémentaires des précédents, sont désignés ports* passifs*.*

Les ports entre lesquels nous allons considérer le délai de séparabilité seront précisément les ports *actifs*. Les ports complémentaires de ces derniers seront les ports *passifs*. Intuitivement, nous serions tentés d'affirmer que les données sur les ports passifs n'influent pas sur le comportement du reste de la machine et d'effacer purement et simplement les données sur les ports passifs. Nous verrons dans la suite que ce n'est pas le cas.

Etant donnés ces ports actifs et passifs, les isomorphismes de commutativité et d'associativité sous lesquels nous travaillons nous permettent d'associer à toute machine $M \in \mathsf{Moore}(In, Out)$ une machine :

$$M' \in \mathsf{Moore}(\mathcal{T}^+_{In} \times \mathcal{T}^-_{In}, \mathcal{T}^+_{Out} \times \mathcal{T}^-_{Out})$$

telle que $M \simeq M'$. Nous dirons que l'interface de la machine M' est *bien structurée*, car elle met en évidence les ports sur lesquels nous voulons déterminer le délai de séparabilité.

Afin d'alléger la notation de la projection de donnée $\prod[In, Out]$, nous omettrons les paramètres In, Out lorsque ces paramètres sont précisément les ports actifs d'une machine. Dans la suite, nous ne traiterons que le cas des machines dont les ports sont bien structurés, sans perte de généralité.

Soit une machine sur treillis à l'interface bien structurée :

$$M = \langle In^+ \times In^-, Out^+ \times Out^-, Q, \mathsf{out}, E, q_0\rangle.$$

Nous définissons un *contexte* comme un mot infini appliqué aux ports passifs d'une machine de Moore depuis un état particulier, suivi d'une restriction de l'interface aux ports actifs. Afin de donner une définition plus synthétique, nous considérons des systèmes sous forme de machines dépliées (cf. Chapitre 2 Sec. 2.2.4 p. 29).

Définition 26 (Application d'un contexte à un état)**.** *Soient $q \in Q$ un état quelconque et $\mathcal{U}_M(q)$ le dépliage de M à partir de q. Soit $cw = a.cw' \in (In^-)^\omega$ un mot infini pour les ports passifs de M. Nous définissons l'application du contexte cw au dépliage $\mathcal{U}_M(q)$, notée $\mathcal{C}(\mathcal{U}_M(q), cw)$, par :*

$$\mathcal{C}(\mathcal{U}_M(q), a.cw') = \left(\mathcal{A}_{Out}(\mathsf{out}(q)), \left\{ (\mathcal{A}_{In}(in), \mathcal{C}(\mathcal{U}_M(q'), cw')) \;\middle|\; \begin{array}{l} q \xrightarrow{in} q' \in E, \\ \mathcal{P}_{In}(in) \sqcap^-_I a \neq \bot_I \end{array} \right\}\right).$$

Cette définition calcule la restriction d'une machine de Moore aux transitions compatibles avec le mot de contexte. Par commodité, pour tout état q on notera $\mathcal{C}(q, cw)$ en lieu et place de $\mathcal{C}(\mathcal{U}_M(q), cw)$.

3.3.1.2 Effacement des données sur les ports passifs comme abstraction

La notion de projection définie précédemment peut être vue comme une fonction d'abstraction de machines. Informellement, le système de transitions étiqueté associé à toute machine M a pour espace d'état le produit cartésien $Q \times Out$, si Q est l'espace d'état de M et Out son

alphabet de sortie. Effacer une partie des données sur Out revient à quotienter l'espace d'état du système de transition étiqueté sous-jacent par la relation d'équivalence induite par la fonction d'effacement. Nous formalisons cette idée dans le cadre des machines sur treillis. Les actions d'effacer les composantes $\mathcal{T}_{In}^-$ et $\mathcal{T}_{Out}^-$ peuvent s'écrire :

$$\begin{array}{lcll} \alpha_{In} & : & \mathcal{T}_{In}^+ \times \mathcal{T}_{In}^- \to \mathcal{T}_{In}^+ & = & \mathcal{A}_{In}, \\ & & & = & \pi_{In^+} \\ \alpha_{Out} & : & \mathcal{T}_{Out}^+ \times \mathcal{T}_{Out}^- \to \mathcal{T}_{Out}^+ & = & \mathcal{A}_{Out}. \\ & & & = & \pi_{Out^+} \end{array}$$

Ces opérations induisent deux correspondances de Galois entre le treillis produit $\mathcal{T}_{In}^+ \times \mathcal{T}_{In}^-$ et $\mathcal{T}_{In}^+$ d'une part et entre $\mathcal{T}_{Out}^+ \times \mathcal{T}_{Out}^-$ et $\mathcal{T}_{Out}^+$ d'autre part. Les fonctions de concrétisation sont les suivantes :

$$\begin{array}{lcl} \gamma_{In} & = & \lambda in.(in, \top_X), \\ \gamma_{Out} & = & \lambda out.(out, \top_Y). \end{array}$$

Il est facile de vérifier que ces fonctions définissent des correspondances, c'est-à-dire que :

$$\begin{array}{l} (\alpha_{In}, \gamma_{In}) \in Galois(\mathcal{T}_{In}^+ \times \mathcal{T}_{In}^-, \mathcal{T}_{In}^+) \\ (\alpha_{Out}, \gamma_{Out}) \in Galois(\mathcal{T}_{Out}^+ \times \mathcal{T}_{Out}^-, \mathcal{T}_{Out}) \end{array}$$

puisque $\alpha(a,b) \preccurlyeq a \iff (a,b) \preccurlyeq (a, \top)$. L'extension aux machines de Moore de cette abstraction de données est précisément l'opération de projection $\prod$ définie plus haut.

La forme de la fonction de concrétisation suggère que plutôt que calculer explicitement une projection, nous pouvons nous contenter des fonctions d'abstraction suivantes :

$$\begin{array}{lcll} \alpha_{In} & : & \mathcal{T}_{In}^+ \times \mathcal{T}_{In}^- \to \mathcal{T}_{In}^+ \times \mathcal{T}_{In}^- & = & \lambda(in,x).(in, \top_X) \\ \alpha_{Out} & : & \mathcal{T}_{Out}^+ \times \mathcal{T}_{Out}^- \to \mathcal{T}_{Out}^+ \times \mathcal{T}_{Out}^- & = & \lambda(out,x).(out, \top_Y). \end{array}$$

Dans les développements qui suivent, nous utiliserons pour plus de clarté la première variante, sauf où mentionné explicitement.

3.3.2 Notions contextuelles de délai de séparabilité et de séparateur

Nous pouvons maintenant procéder à l'extension des définitions sur le délai de séparabilité au cadre de machines dotées de multiples ports. Il existe au moins deux façons de généraliser le délai de séparabilité, correspondant à quantifier universellement ou existentiellement sur les données provenant des ports passifs. Soit $M \in \mathsf{Moore}(\mathcal{T}_{In}^+ \times \mathcal{T}_{In}^-, \mathcal{T}_{Out}^+ \times \mathcal{T}_{Out}^-)$ une machine telle que définie précédemment.

Définition 27 (Délai de séparabilité contextuel)**.** *Soit $q \in Q$ un état. Le délai de séparabilité optimiste de q entre les ports $\mathcal{T}_{In}^+$ et $\mathcal{T}_{Out}^+$ est calculé comme le minimum sur tous les contextes du délai de séparabilité optimiste.*

$$\mathsf{septime}_{\mathsf{O}}^{\alpha}(\mathcal{T}_{In}^+, \mathcal{T}_{Out}^+, q) = \min\left\{\mathsf{septime}_{\mathsf{O}}^{\alpha}(\mathcal{C}(q,cw)) \mid cw \in (In^-)^\omega\right\}.$$

Le délai de séparabilité pessimiste entre les ports $\mathcal{T}_{In}^+$ et $\mathcal{T}_{Out}^+$ est calculé de façon duale, comme le maximum sur tous les contextes du délai de séparabilité pessimiste :

$$\mathsf{septime}_{\mathsf{P}}^{\alpha}(\mathcal{T}_{In}^+, \mathcal{T}_{Out}^+, q) = \max\left\{\mathsf{septime}_{\mathsf{P}}^{\alpha}(\mathcal{C}(q,cw)) \mid cw \in (In^-)^\omega\right\}.$$

Les séparateurs se contextualisent de la même façon que le délai de séparabilité. Soient $q_1, q_2 \in Q$ deux états. Nous définissons une notion contextuelle de séparateur .

Définition 28 (Séparateur dans un contexte particulier)**.** *Soit $cw \in (In^-)^\omega$ un contexte. L'ensemble des séparateurs de $(q_1, q_2) \in Q \times Q$ dans le contexte cw est :*

$$\mathcal{C}(\mathsf{S}^\alpha(q_1, q_2), cw) = \{w \mid w \in \mathsf{S}^\alpha(\mathcal{C}(q_1, cw), \mathcal{C}(q_2, cw))\}.$$

Les ensembles des séparateurs effectifs et des pseudo-séparateurs dans le contexte cw (resp. $\mathcal{C}(\mathsf{S}^\alpha_!(q_1, q_2), cw)$ et $\mathcal{C}(\mathsf{S}^\alpha_*(q_1, q_2), cw)$) sont définis similairement.

Les *séparateurs existentiels* sont ceux dont la propriété d'être des séparateurs n'est valide que dans certains contextes. La notion de *séparateur universel* correspond aux mots qui conservent la propriété d'être des séparateurs dans tous les contextes.

Définition 29 (Séparateurs existentiels et universels)**.** *L'ensemble des séparateurs existentiels de $(q_1, q_2) \in Q \times Q$ est $\exists\text{-}\mathsf{S}^\alpha(q_1, q_2)$ et est défini par :*

$$\exists\text{-}\mathsf{S}^\alpha(q_1, q_2) = \{w \mid \exists cw, w \in \mathcal{C}(\mathsf{S}^\alpha(q_1, q_2), cw)\}.$$

L'ensemble des séparateurs universels de (q_1, q_2) est noté $\forall\text{-}\mathsf{S}^\alpha(q_1, q_2)$ et est défini par :

$$\forall\text{-}\mathsf{S}^\alpha(q_1, q_2) = \{w \mid \forall cw, w \in \mathcal{C}(\mathsf{S}^\alpha(q_1, q_2), cw)\}.$$

Les ensembles des séparateurs effectifs ($\exists\text{-}\mathsf{S}^\alpha_!$, $\forall\text{-}\mathsf{S}^\alpha_!$) et des pseudo-séparateurs ($\exists\text{-}\mathsf{S}^\alpha_*$, $\forall\text{-}\mathsf{S}^\alpha_*$) sont définis de façon similaire.

3.3.3 Force relative des séparateurs

Cas des séparateurs. Afin d'obtenir une méthode de vérification efficace, il est utile d'étudier la force relative de ces définitions, afin d'établir des relations de sur et sous-approximation. Nous comparons également les définitions ci-dessus à l'action d'effacer les données sur les ports passifs. Il se trouve que dans le cas des séparateurs non-effectifs, il n'existe pas de lien clair avec l'effacement. Soit $M \in \mathsf{Moore}(\mathcal{T}^+_{In} \times \mathcal{T}^-_{In}, \mathcal{T}^+_{Out} \times \mathcal{T}^-_{Out})$ une machine telle que définie précédemment :

$$M = \langle In^+ \times In^-, Out^+ \times Out^-, Q, \mathsf{out}, E, q_0 \rangle.$$

La proposition suivante décrit les relations d'inclusions entre les sensembles de séparateurs existentiels, universels et après effacement.

Proposition 3.20 (Force relative des séparateurs)**.** *Soient $(q_1, q_2) \in Q \times Q$. Posons les projections sur les ports actifs :*

$$proj_1 = \prod(\mathcal{U}_M(q_1)),$$
$$proj_2 = \prod(\mathcal{U}_M(q_2)).$$

L' inclusion suivante est vérifiée :

$$\forall w \in \mathsf{S}^\alpha(proj_1, proj_2), \exists cw, w \in \mathcal{C}(\mathsf{S}^\alpha(q_1, q_2), cw).$$

De plus, $\forall\text{-}\mathsf{S}^\alpha(q_1, q_2)$ et $\mathsf{S}^\alpha(proj_1, proj_2)$ sont incomparables.

Cette proposition affirme que les séparateurs calculés après projection sur-approximent les séparateurs existentiels. Aucun lien avec les séparateurs universels n'apparait.

Démonstration.

■ Nous prouvons $\forall w \in \mathsf{S}^\alpha(proj_1, proj_2), \exists cw, w \in \mathcal{C}(\mathsf{S}^\alpha(q_1, q_2), cw)$.
Soit $w \in \mathsf{S}^\alpha(proj_1, proj_2)$. Ceci revient à écrire :

$$w \in \mathsf{S}^\alpha(\prod(q_1), \prod(q_2)).$$

Nous procédons par induction sur w pour montrer l'existence d'un contexte cw permettant de séparer (q_1, q_2).

- Le cas $w = \epsilon$ est direct, puisque tous les mots infinis sont des contextes valides.
- Par utilisation de la définition des séparateurs sur $w = a^+.w'$, nous avons :

$$\exists q'_i \in Q, \exists q_i \xrightarrow{a^+} q'_i \in \prod(E), \forall q'_j \in Q, \forall a_j \preccurlyeq_I a^+, q_j \xrightarrow{a_j} q'_j \in \prod(E) \implies \mathsf{c\text{-}ex}^\alpha(q'_i, q'_j)$$

$$((i,j) = (1,2) \vee (i,j) = (2,1)).$$

Par définition de l'effacement, nous obtenons de la proposition précédente :

$$\exists a^- \in In^-, \exists q_i \xrightarrow{\langle a^+, a^- \rangle} q'_i \in E, \forall q'_j \in Q, \forall a_j \preccurlyeq a^+, \\ q_j \xrightarrow{\langle a_j, \_ \rangle} q'_j \in E \implies \mathsf{c\text{-}ex}^\alpha(\prod(q'_i), \prod(q'_j))$$

$$((i,j) = (1,2) \vee (i,j) = (2,1))$$

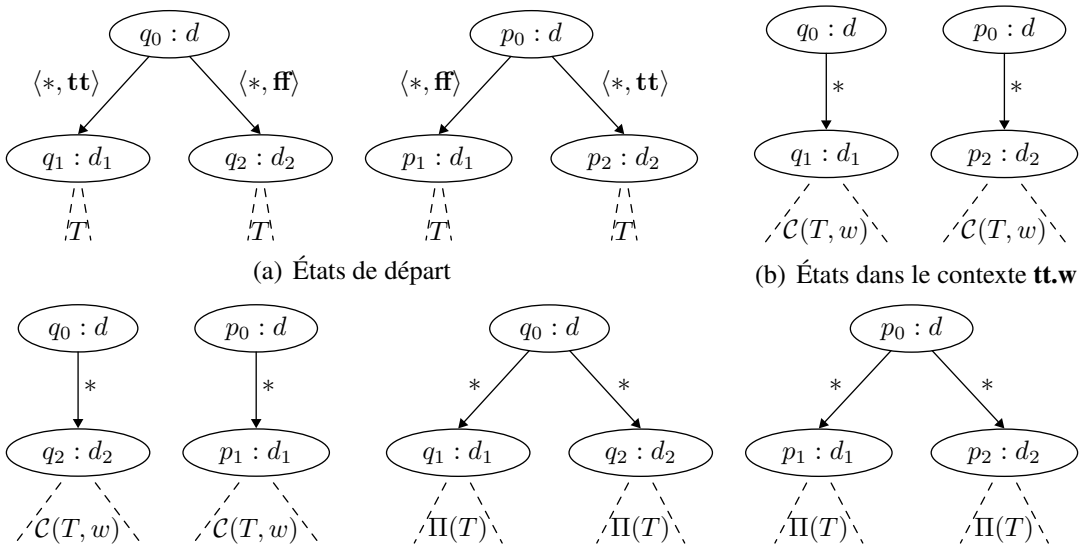
Par hypothèse d'induction, il existe cw' tel que $w' \in \mathcal{C}(\mathsf{S}^\alpha(q'_1, q'_2), cw')$ et nous trouvons $cw = a^-.cw'$.

■ Le fait que $\forall\text{-}\mathsf{S}^\alpha(q_1, q_2) \notin \mathsf{S}^\alpha(proj_1, proj_2)$ est montré par le contre-exemple en Fig. 3.7. Nous utilisons des machines concrètes, mais le résultat s'étend directement au cas des machines abstraites. Ce contre exemple montre deux machines d'états initiaux respectivement q_0 et p_0 et de signature $\mathsf{Moore}(X^+ \times \mathbf{bool}^-, Y^+)$, où X et Y sont des ensembles quelconques et où $d_1 \neq d_2$. Ces états montrent l'existence d'un ensemble de séparateurs universels (en l'occurrence, $\forall\text{-}\mathsf{S}^\alpha(q_1, q_2) = X$) qui ne sont pas conservés par projection.

■ Enfin, le fait que $\mathsf{S}^\alpha(proj_1, proj_2) \notin \forall\text{-}\mathsf{S}^\alpha(q_1, q_2)$ est montré par le contre-exemple en Fig. 3.8, qui montre deux machines telles qu'un séparateur (ici, **tt**) sur les projections ne soit pas un séparateur universel. En l'occurrence, **tt** n'est pas un séparateur de q_0, p_0 dans le contexte $1.w$ où w est quelconque.

□

La proposition précédente s'intéresse au cas général. Étudions maintenant le cas particulier des séparateurs effectifs. Nous observons que les séparateurs des machines de Moore après effacement sont inclus dans les séparateurs effectifs universels, ce qui nous permettra de sous-approximer ces derniers efficacement.



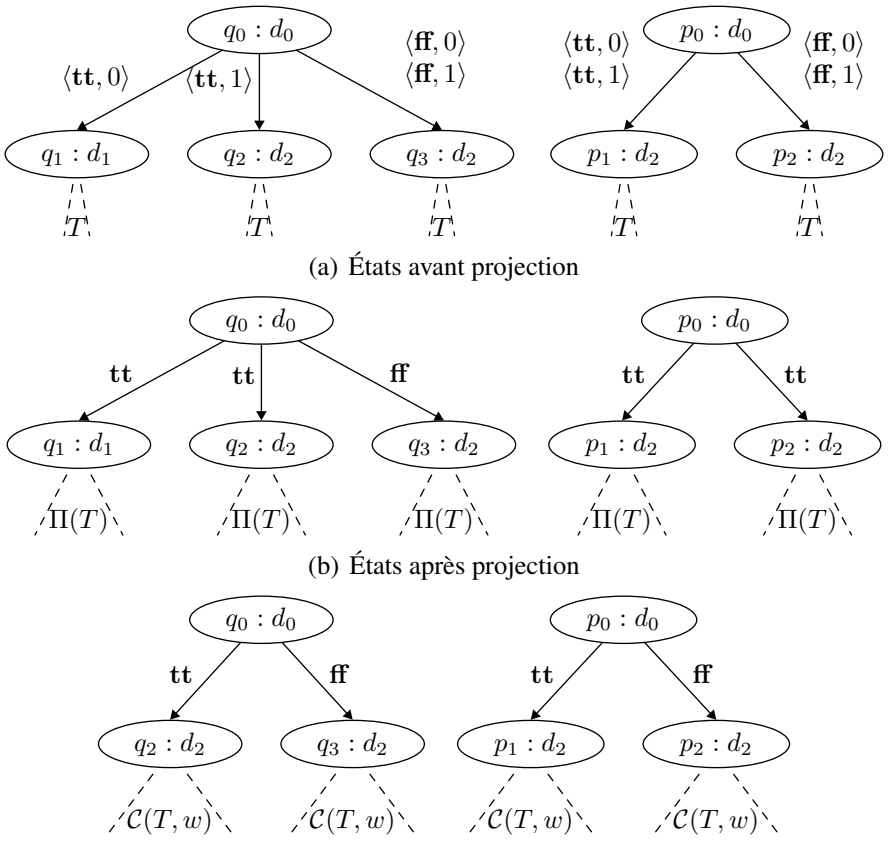
(a) États de départ

(b) États dans le contexte **tt.w**

(c) États dans le contexte **ff.w**

(d) États après projection

FIGURE 3.7 – Contre-exemple à $\forall\text{-}\mathsf{S}^{\alpha}(q_1, q_2) \subseteq \mathsf{S}^{\alpha}(proj_1, proj_2)$.



(a) États avant projection

(b) États après projection

(c) États dans le contexte **1.w**

FIGURE 3.8 – Contre-exemple à $\mathsf{S}^{\alpha}(proj_1, proj_2) \subseteq \forall\text{-}\mathsf{S}^{\alpha}(q_1, q_2)$.

Proposition 3.21 (Force relative des séparateurs effectifs)**.** *L' inclusion suivante est vérifiée :*

$$\mathsf{S}^{\alpha}_{!}(proj_1, proj_2) \subseteq \forall\text{-}\mathsf{S}^{\alpha}_{!}(q_1, q_2).$$

Démonstration. Nous prouvons l'inclusion $\mathsf{S}^{\alpha}_{!}(proj_1, proj_2) \subseteq \forall\text{-}\mathsf{S}^{\alpha}_{!}(q_1, q_2)$ par induction, de la même façon que précédemment. Le cas de base $w = \epsilon$ est direct. Dans le cas inductif, nous avons $w = a^{+}.w' \in \mathsf{S}^{\alpha}_{!}(proj_1, proj_2)$. Par définition des séparateurs effectifs, pour tous $a_1, a_2 \preccurlyeq_I a^{+}$,

$$\forall q_1' \in Q, q_2' \in Q, q_1 \xrightarrow{a_1} q_1' \in \prod(E) \wedge q_2 \xrightarrow{a_2} q_2' \in \prod(E) \implies \mathsf{c\text{-}ex}^{\alpha}(\prod(q_1'), \prod(q_2')).$$

Par définition de l'effacement, nous avons :

$$\forall x, y \in In^{-}, q_1 \xrightarrow{\langle a_1, x\rangle} q_1' \in \prod(E) \ \wedge \ q_2 \xrightarrow{\langle a_2, y\rangle} q_2' \in \prod(E) \implies \mathsf{c\text{-}ex}^{\alpha}\left(\prod(q_i'), \prod(q_j')\right).$$

Soit $cw = a^{-}.cw'$ un contexte quelconque. En fixant $x = y = a^{-}$, nous trouvons :

$$q_1 \xrightarrow{\langle a_1, a^{-}\rangle} q_1' \in \prod(E) \wedge q_2 \xrightarrow{\langle a_2, a^{-}\rangle} q_2' \in \prod(E) \implies \mathsf{c\text{-}ex}^{\alpha}\left(\prod(q_i'), \prod(q_j')\right).$$

Par hypothèse d'induction, $w' \in \forall\text{-}\mathsf{S}^{\alpha}_{!}(q_1', q_2')$. En utilisant la propriété ci-dessus, nous concluons $w \in \forall\text{-}\mathsf{S}^{\alpha}_{!}(q_1, q_2)$. □

Le cas des séparateurs effectifs est donc plus intéressant : il est possible de sous-approximer les séparateurs effectifs universels en calculant les séparateurs effectifs après effacement. Cela nous permet d'éviter de calculer l'intersection abstraite de tous les séparateurs effectifs pour tous les contextes.

Cas des pseudo-séparateurs. Les pseudo-séparateurs ne vérifient pas de chaîne d'inclusion semblable aux séparateurs simples, mais nous pouvons montrer une propriété plus faible : si après effacement il existe un pseudo-séparateur w de deux états, alors ces deux états génèrent bien des langages disjoints pour ce même mot w mais w peut perdre la propriété d'être de taille minimale et donc d'être un pseudo-séparateur. Ceci est exprimé dans la proposition suivante.

Proposition 3.22 (Force relative des pseudo-séparateurs)**.** *Les pseudo-séparateurs après projection garantissent des langages disjoints pour tous les contextes sur les états avant projection.*

$$\exists w \in \mathsf{S}^{\alpha}_{*}(proj_1, proj_2) \rightarrow \forall cw, \exists w' \in prefix(w), w' \in Disj(\mathcal{C}(q_1, cw), \mathcal{C}(q_2, cw)).$$

Démonstration. Nous raisonnons par l'absurde. Posons comme hypothèse $w \in \mathsf{S}^{\alpha}_{*}(proj_1, proj_2)$. Soit cw un contexte tel que pour tout préfixe w' de w, w' induise des langages non-disjoints. Nous pouvons en particulier poser $w' = w$. Ceci est encore équivalent à l'existence d'une séquence d'exécution induite par w générant le même mot de sortie. En effectuant l'opération de projection, cette séquence est conservée ce qui entraîne $w \notin \mathsf{S}^{\alpha}_{*}(proj_1, proj_2)$. De cette contradiction, nous déduisons la proposition initiale. □

3.3.4 Force relative des paires séparantes

Nous appliquons une méthodologie semblable à celle utilisée pour étudier la force relative des séparateurs afin de caractériser les propriétés conservées par les paires séparantes lors de l'opération d'effacement. Comme dans le cas des séparateurs, nous définissons les paires séparantes en modulant la quantification sur les contextes et nous étudions pour chaque variante si ces paires sont conservées par effacement. Nous travaillerons sur une machine à l'interface bien structurée $M \in \mathsf{Moore}(\mathcal{T}^{+}_{In} \times \mathcal{T}^{-}_{In}, \mathcal{T}^{+}_{Out} \times \mathcal{T}^{-}_{Out})$ comme décrite précédemment.

3.3.4.1 Cas des paires séparantes simples

Nous pouvons sous-approximer les paires séparantes dans le cas existentiel en travaillant après effacement. Ce fait est une conséquence du résultat de sous-approximation des séparateurs potentiels.

Proposition 3.23. *Soit $q \in Q$ un état et soit $(a_1, a_2) \in \mathsf{SepPairs}^{\alpha}(\prod(q))$. Alors il existe un contexte cw tel que $(a_1, a_2) \in \mathsf{SepPairs}^{\alpha}(\mathcal{C}(q, cw))$.*

Démonstration. Construisons un contexte cw permettant de pouver la proposition. Nous avons par définition de $\mathsf{SepPairs}^{\alpha}(\prod(q))$:

$$\exists q_i \in Q, \exists q \xrightarrow{a_i} q_i \in \prod(E), \forall b_j \preccurlyeq_I a_j, q \xrightarrow{b_j} q_j \in \prod(E) \implies \mathsf{c{-}ex}^{\alpha}(\prod(q_i), \prod(q_j))$$

$$((i,j) = (1,2) \vee (i,j) = (2,1))$$

Par définition de la projection,

$$\exists a^{-}, \exists q_i \in Q, \exists q \xrightarrow{\langle a_i, a^{-}\rangle} q_i \in E, \forall b_j \preccurlyeq_I a_j, q \xrightarrow{\langle b_j, \_\rangle} q_j \in E \implies \mathsf{c{-}ex}^{\alpha}(\prod(q_i), \prod(q_j)).$$

Par la proposition 3.20, il existe un contexte $cw'_{i,j}$ tel que :

$$\mathsf{c{-}ex}^{\alpha}(\mathcal{C}(q_i, cw'_{i,j}), \mathcal{C}(q_j, cw'_{i,j})).$$

Nous en déduisons l'existence d'un ensemble de contextes :

$$\left\{a^{-}.cw'_{i,j} \mid (a_1, a_2) \in \mathsf{SepPairs}^{\alpha}(\mathcal{C}(q, cw))\right\}.$$

□

Nous avons montré plus haut que les séparateurs potentiels universels ne sont pas comparables avec les séparateurs potentiels sur projection. Ce résultat négatif se transfère aux paires séparantes universelles.

Proposition 3.24. *Il existe une machine abstraite M dotée d'un état q tel que la propriété suivante soit vérifiée :*

$$\exists (a_1, a_2) \in \mathsf{SepPairs}^{\alpha}(\prod(q)), \exists cw, (a_1, a_2) \notin \mathsf{SepPairs}^{\alpha}(\mathcal{C}(q, cw)).$$

Démonstration. La preuve suit directement du contre-exemple montré en Fig. 3.8. p. 78. En effet, si une paire d'états (q_1, q_2) peut perdre des séparateurs par projection, elle peut également perdre la propriété d'être séparable. □

3.3.4.2 Cas des paires séparantes stables

Dans le cas abstrait, la définition des paires séparantes stables est similaire à sa contrepartie concrète (Def. 14 p. 46). Nous rappellons que les paires séparantes stables sont celles qui ont la propriété d'être conservées par simulation inverse (i.e. concrétisation).

Définition 30 (Paires séparantes abstraites stables)**.** *Soit* $M \in \mathsf{Moore}(\mathcal{T}_{In}, \mathcal{T}_{Out})$ *et* q *un état de* M*. La paire* $(a_1, a_2) \in \mathcal{T}_{In} \times \mathcal{T}_{In}$ *est une paire séparante abstraite* stable *si et seulement si :*

$$\forall b_1 \preccurlyeq_I a_1, \forall b_2 \preccurlyeq_I a_2, \forall q_1, q_2 \in Q, q \xrightarrow{b_1} q_1 \ \wedge \ q \xrightarrow{b_2} q_2 \implies \exists w \in \mathsf{S}^{\alpha}_{*}(q_1, q_2).$$

L'ensemble des paires séparantes stables de q *est noté* $\mathsf{SepPairs}^{\alpha}_{*}(q)$.

Une propriété des paires séparantes stables est qu'elles sont sous-approximées par l'opération d'effacement sur les machines, ce qui ouvre la possibilité de s'en servir après abstraction.

Proposition 3.25. *Soit* $q \in Q$ *un état et soit* $(a_1, a_2) \in \mathsf{SepPairs}^{\alpha}_{*}(\prod(q))$*. Alors pour tout contexte* cw*,* $(a_1, a_2) \in \mathsf{SepPairs}^{\alpha}(\mathcal{C}(q, cw))$.

Démonstration. Considérons la définition des paires séparantes stables donnée plus haut. Posons $b_1 \preccurlyeq_I a_1$, $b_2 \preccurlyeq_I a_2$, $q_1, q_2 \in Q$ quelconques. Par définition de l'effacement, nous obtenons :

$$\forall x, y \in In^{-}, q \xrightarrow{\langle b_1, x \rangle} q_1 \in \prod(E) \wedge q \xrightarrow{\langle b_2, y \rangle} q_2 \in \prod(E) \implies \exists w \in \mathsf{S}^{\alpha}_{*}(\prod(q_1), \prod(q_2)).$$

Pour tout séparateur $w \in \mathsf{S}^{\alpha}_{*}(\prod(q_1), \prod(q_2))$, la proposition 3.22 nous permet d'affirmer l'existence d'un préfixe $w' \in prefix(w)$ induisant des langages disjoints et donc d'un pseudo-séparateur $w_* \in \mathsf{S}^{\alpha}_{*}(\mathcal{C}(q_1, cw'), \mathcal{C}(q_2, cw'))$. Pour tout contexte $cw = a^{-}.cw'$, nous pouvons donc écrire :

$$q \xrightarrow{\langle b_1, a^{-} \rangle} q_1 \in E \ \wedge \ q \xrightarrow{\langle b_2, a^{-} \rangle} q_2 \in E \implies \exists w_* \in \mathsf{S}^{\alpha}_{*}(\mathcal{C}(q_1, cw), \mathcal{C}(q_2, cw)).$$

Ceci nous permet de conclure que $\forall cw, (a_1, a_2) \in \mathsf{SepPairs}^{\alpha}(\mathcal{C}(q, cw))$. □

Dans cette section, nous avons apporté un ensemble de résultats de conservation et de non-conservation de propriétés de séparabilité en fonction des deux paramètres suivants :

- le type de quantification sur les contextes,
- l'opération d'effacement des données sur les ports passifs.

3.3.5 Délai de séparabilité et effacement

Les propositions précédentes nous permettent d'énoncer certains résultats d'approximation du délai de séparabilité par effacement. Lorsque c'est possible, ceci nous offre la possibilité de réduire la complexité de l'analyse du délai de séparabilité de façon proportionnelle à la réduction de la taille de l'espace d'états. Les résultats présentés dans cette section sont donc d'une importance centrale.

3.3.5.1 Délai de séparabilité optimiste

Dans le cas optimiste, il est possible de donner une estimation conservative du délai de séparabilité en procédant à l'effacement des données sur les ports passifs.

Théorème 3.26 (Sur-approximation du délai de séparabilité optimiste par effacement)**.**

$$\mathsf{septime}^{\alpha}_{\mathsf{O}}(\mathcal{T}^{+}_{In}, \mathcal{T}^{+}_{Out}, q) \leq \mathsf{septime}^{\alpha}_{\mathsf{O}}(\prod(q)).$$

Ce théorème découle informellement des éléments suivants.
- L'ensemble des paires séparantes après effacement est *inclus* dans celles avant effacement ;
- pour chaque paire séparante, les états atteints après effacement sont un *sous-ensemble* de ceux atteints avant effacement ;
- pour chacune de ces paires d'états, les séparateurs après effacement sont à nouveaux un sous-ensemble des séparateurs avant effacement.

Par conséquent, le délai de séparabilité après effacement est supérieur ou égal à celui avant effacement.

Démonstration. Le délai contextuel optimiste est calculé comme un minimum sur tous les contextes. Nous devons donc montrer qu'il existe un contexte $cw \in (In^-)^\omega$ tel que :

$$\mathsf{septime}^\alpha_{\mathsf{O}}(\mathcal{C}(q, cw)) \leq \mathsf{septime}^\alpha_{\mathsf{O}}(\prod(q)).$$

En appliquant les définitions de délai de séparabilité, le but se réécrit en :

$$\exists cw, \mathsf{septime}^\alpha_{\mathsf{O}}(\mathcal{C}(q, cw)) \leq \mathsf{min} \left\{ \mathsf{septime}^\alpha_{\mathsf{O}}(q_1, q_2) \;\middle|\; \begin{array}{l} \exists (a_1, a_2) \in \mathsf{SepPairs}^\alpha(\prod(q)), \\ \exists b_1 \preccurlyeq_I a_1, b_2 \preccurlyeq_I a_2, \\ q \xrightarrow{b_1} q_1 \in \prod(E) \;\wedge\; q \xrightarrow{b_2} q_2 \in \prod(E) \end{array} \right\}$$

Considérons deux transitions $q \xrightarrow{b_1} q_1$ et $q \xrightarrow{b_2} q_2$ telles que $\mathsf{c{-}ex}^\alpha(\prod(q_1), \prod(q_2))$ soit vérifié. Soit $w \in \mathsf{S}^\alpha(\prod(q_1), \prod(q_2))$ le séparateur de (q_1, q_2) de taille minimale *après effacement*. Par la proposition 3.20 p. 76 :

$$\exists cw' \in \mathsf{S}^\alpha(\mathcal{C}(q_1, cw'), \mathcal{C}(q_2, cw')).$$

Nous avons donc $\mathsf{septime}^\alpha_{\mathsf{O}}(\mathcal{C}(q_1, cw'), \mathcal{C}(q_2, cw')) \leq \mathsf{septime}^\alpha_{\mathsf{O}}(\prod(q_1), \prod(q_2))$. Par utilisation de la proposition 3.23 p. 80, il existe un ensemble de contextes :

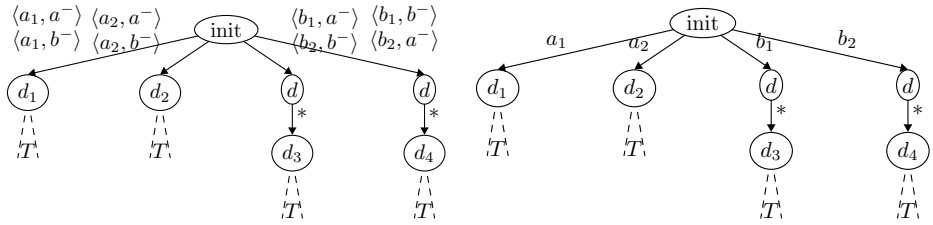
$$\{a^-.cw' \mid (a_1, a_2) \in \mathsf{SepPairs}^\alpha(\mathcal{C}(q, a^- cw'))\},$$

ce qui achève de prouver le but. □

3.3.5.2 Délai de séparabilité pessimiste

De façon duale au délai de séparabilité contextuel optimiste, le délai contextuel pessimiste est calculé comme le maximum du délai de séparabilité pessimiste pour tous les contextes. De ce fait, l'abstraction consistant à effacer les données sur les ports passifs ne permet pas de calculer de sur-approximation du délai contextuel pessimiste. Le contre-exemple en Fig. 3.9 montre initialement un dépliage dans lequel $d_1 \neq d_2$ et $d_3 \neq d_4$. Les données sur les ports passifs sont notées a^- et b^-. Avant effacement, les paires séparantes sont $\{(a_1, a_2); (b_1, b_2)\}$ dans tous les contextes. Après effacement, la paire (b_1, b_2) n'est plus séparante car les états atteignables par b_1 et b_2 sont les mêmes. Le délai de séparabilité après effacement ne tient pas compte de cette paire et est égal à 0, alors que le maximum sur tous les contextes du délai de séparabilité pessimiste est égal à 1.

Le problème de l'abstraction proposée initialement est qu'elle sous-approxime l'ensemble des paires séparantes. Puisque le délai optimiste est calculé comme un minimum sur un ensemble de délais de séparabilité de paires séparantes, sous-approximer cet ensemble calcule bien



(a) Avant effacement (b) Après effacement

FIGURE 3.9 – Non-conservation du délai de séparabilité pessimiste

une sur-approximation du délai de séparabilité optimiste de l'état. Cependant, ceci n'est pas applicable au délai pessimiste, qui procède par le calcul d'un *maximum*. Nous devons amender notre méthode d'abstraction afin de tenir compte de cela. La méthode que nous proposons consiste en ne pas effacer les données des ports passifs sur les arcs directement successeurs de l'état dont nous voulons approximer le délai de séparabilité pessimiste, ce qui revient à ne pas approximer les paires séparantes.

Théorème 3.27 (Sur-approximation du délai de séparabilité pessimiste par effacement partiel)**.** *Soit* $M = \langle \mathcal{T}^+_{In} \times \mathcal{T}^-_{In}, \mathcal{T}^+_{Out} \times \mathcal{T}^-_{Out}, Q, \mathsf{out}, E, q_0 \rangle$ *une machine sur treillis et soit* $q \in Q$ *un état. Nous définissons son* effacement partiel*, noté* $\tilde{\prod}[\mathcal{T}^+_{In}, \mathcal{T}^+_{Out}]$ *sur son dépliage* $\mathcal{U}(q)$ *:*

$$\tilde{\prod}[\mathcal{T}^+_{In}, \mathcal{T}^+_{Out}](\mathcal{U}(q)) = \left(\pi_{\mathcal{T}^+_{Out}}(\mathsf{out}(q)), \left\{(in, \prod(\mathcal{U}(q'))) \mid q \xrightarrow{in} q' \in E\right\}\right).$$

L'effacement partiel vérifie la propriété suivante :

$$\mathsf{septime}^\alpha_{\mathsf{P}}(\mathcal{T}^+_{In}, \mathcal{T}^+_{Out}, q) \leq \mathsf{septime}^\alpha_{\mathsf{P}}(\tilde{\prod}[\mathcal{T}^+_{In}, \mathcal{T}^+_{Out}](q))$$

Démonstration. Considérons une paire séparante $(a_1, a_2) \in \mathsf{SepPairs}^\alpha(\mathcal{C}(q, cw))$ dans un contexte quelconque $cw = a^-.cw'$ et montrons que son délai est majoré par une paire séparante après projection. Puisque le délai est fini, nous avons nécessairement $(a_1, a_2) \in \mathsf{SepPairs}^\alpha_*(\mathcal{C}(q, cw))$:

$$q \xrightarrow{\langle a_1, a^- \rangle} q_1 \in E \ \wedge \ q \xrightarrow{\langle a_2, a^- \rangle} q_2 \in E \implies \mathcal{C}(q_1, cw') \ )(^\alpha \ \mathcal{C}(q_2, cw').$$

Le délai de séparabilité de (a_1, a_2) est égal à :

$$\mathsf{max}\{|w| \mid w \in \mathsf{S}^\alpha_*(\mathcal{C}(q_1, cw'), \mathcal{C}(q_2, cw'))\},$$

où q_1 et q_2 sont atteignables par (a_1, a_2). Soit $w_{max} \in \mathsf{S}^\alpha_*(\mathcal{C}(q_1, cw'), \mathcal{C}(q_2, cw'))$ le pseudo-séparateur de longueur maximale pour q dans le contexte cw. Nous procédons par analyse par cas sur la valeur de $\mathsf{septime}^\alpha_{\mathsf{P}}(\tilde{\prod}[\mathcal{T}^+_{In}, \mathcal{T}^+_{Out}](q))$.

- Si il est infini, alors la preuve est directement terminée.
- Sinon, nous avons par définition de l'effacement partiel sur q :

$$q \xrightarrow{\langle a_1, a^- \rangle} q_1 \in E \ \wedge \ q \xrightarrow{\langle a_2, a^- \rangle} q_2 \in E \implies \prod(q_1) \ )(^\alpha \ \prod(q_2).$$

  Considérons à nouveau le délai de séparabilité de q_1 et q_2, cette fois ci après effacement. Considérons l'ensemble $W = \{w_{max}.suffix \mid suffix \in (In^+)^\omega\}$ des mots infinis préfixés par w_{max}. Par définition de $)(^\alpha$, il existe pour tout $w_{ext} \in W$ un préfixe w^*_{ext} tel que :

$$w^*_{ext} \in \mathsf{S}^\alpha_*(\prod(q_1), \prod(q_2)).$$

La proposition 3.22 p. 79 nous permet d'affirmer l'existence pour tout w^*_{ext} d'un préfixe $w^*_{pre} \in prefix(w^*_{ext})$ tel que $w^*_{pre} \in \mathsf{S}^{\alpha}_{*}(\mathcal{C}(q_1, cw'), \mathcal{C}(q_2, cw'))$. Procédons à nouveau à une analyse par cas sur la longueur de w^*_{pre}.

– Si $|w^*_{pre}| < |w_{max}|$, alors nous avons une contradiction avec le fait que

$$w_{max} \in \mathsf{S}^{\alpha}_{*}(\mathcal{C}(q_1, cw'), \mathcal{C}(q_2, cw')).$$

– Sinon, nous avons $|w^*_{pre}| \geq |w_{max}|$ ce qui nous permet d'affirmer que le délai de séparabilité après effacement majore bel et bien le délai de séparabilité avant effacement.

□

Ces résultats nous donnent quelques moyens de calculer des approximations de délai de séparabilité en nous restreignant au cas non-contextuel, plus simple. Cependant, nous avons vu que l'abstraction utilisée pour sur-approximer le délai de séparabilité pessimiste est imparfaite, du fait que l'effacement sous-approxime les paires séparantes. Nous résolvons le problème en n'approximant pas les paires séparantes de l'état considéré. Il faut noter que toute sur-approximation de l'ensemble des paires séparantes est une solution valide. D'un autre coté, la portée de ce résultat est diminuée par le fait que le délai de séparabilité pessimiste n'est pas conservé par concrétisation (Sec. 3.2.4.2 p. 70).

3.4 Raffinement

La complexité des systèmes que nous voulons développer motive la création d'une méthodologie de conception incrémentale. Chaque étape du développement doit permettre d'apporter des preuves de correction, ou tout au moins des preuves de cohérence avec la spécification. Ces étapes élémentaires doivent pouvoir exprimer deux axes de développement : l'ajout de détails algorithmiques sur le fonctionnement de chaque machine, et la structuration du système global en un réseau de machines communicantes. Cette section montre comment utiliser les outils formels définis jusqu'à présent pour définir le raffinement. Nous allons décrire chacune des composantes de notre formalisme : les "langages" de spécification et d'implémentation, la relation de satisfaction et les relations de raffinement proprement dites.

3.4.1 Langage de spécification et d'implémentation

Nous choisissons les machines de Moore abstraites comme moyens d'exprimer à la fois spécification et implémentation. Ce procédé est commun à d'autres formalismes basés sur d'autres modèles de machines à états [45]. Cependant, nos machines de spécification voient leurs états enrichis de contraintes de délai de séparabilité. Dans la suite, nous supposons l'existence d'une machine $M = \langle \mathcal{T}_{In}, \mathcal{T}_{Out}, Q, \mathsf{out}, E, q_0 \rangle$.

Définition 31 (Contrainte de délai de séparabilité)**.** *Nous avons défini en Sec. 3.3.1.1 p. 72 la notion de* ports *d'entrée-sortie. Une contrainte de délai de séparabilité est un triplet* $\langle I, O, t, q \rangle$ *où :*

– I *est un port d'entrée ;*
– O *est un port de sortie ;*
– $t \in \mathbb{N}$ *est un entier,*
– $q \in Q$ *est un état de* M.

Une telle contrainte signifiera qu'une implémentation *de q doit avoir sur les ports désignés un délai de séparabilité abstrait (optimiste ou pessimiste, selon la méthode choisie) inférieur ou égal à t. Soit* $\mathsf{Constraints}_M$ *l'ensemble des contraintes de délai de séparabilité sur M.*

Les contraintes de délai de séparabilité nous permettent de construire un exemple de langage de spécification.

Définition 32 (Spécification et implémentation)**.**

1. *Puisque le délai de séparabilité est calculable, la satisfaction d'une contrainte de délai de séparabilité est un problème décidable. Nous pouvons donc construire un calcul propositionnel ayant pour atomes des contraintes de délai de séparabilité. L'ensemble des formules ϕ_M est défini classiquement, comme une fermeture par conjonction et disjonction (d'autres opérateurs sont bien sûr possibles) :*

$$\phi_M ::= \mathsf{Constraints}_M \mid \phi_0 \wedge \phi_1 \mid \phi_0 \vee \phi_1.$$

   Une spécification est un couple $\langle M, spec\rangle$ où $spec \in \phi_M$. La composante M est la spécification fonctionnelle *et la composante $spec$ est la spécification* temporelle*.*
2. *Une implémentation de $\langle M, spec\rangle$ sera une simple machine $M' \in \mathsf{Moore}(\mathcal{T}_{In}, \mathcal{T}_{Out})$ satisfaisant les spécifications. La relation de satisfaction est décrite ci-après.*

3.4.2 Relations de satisfaction

Une relation de satisfaction à pour but de lier une spécification à un ensemble d'implémentations la respectant. Comme défini plus haut, dans notre cadre une spécification est un couple $\langle M, \phi\rangle$ où M est la spécification *fonctionnelle* et $spec$ est la spécification temporelle. Ces deux spécifications n'ont pas le même statut : nous ne contraignons pas la spécification fonctionnelle à respecter la composante temporelle. De ce fait, la relation de satisfaction peut être exprimée d'une part comme une relation de satisfaction fonctionnelle et d'autre part comme une relation de satisfaction temporelle. La spécification et la vérification de propriétés fonctionnelles n'étant pas le coeur de notre sujet, nous ne ferons dans ce manuscrit que proposer une solution classique, la simulation, sans étudier en détail ses propriétés (cela sera fait indirectement lors de l'étude du raffinement). Nous nous concentrerons alors sur la relation de satisfaction temporelle.

Soient $\mathcal{T}_{In}, \mathcal{T}_{Out}, \mathcal{T}'_{In}, \mathcal{T}'_{Out} \in \mathfrak{D}^{\alpha}$ des abstractions de types de données telles qu'il existe :

$$g_{In} \in Galois(\mathcal{T}'_{In}, \mathcal{T}_{In}), \quad g_{Out} \in Galois(\mathcal{T}'_{Out}, \mathcal{T}_{Out}).$$

Définition 33 (Relation de satisfaction fonctionnelle)**.** *Soit $\langle M, spec\rangle$ une spécification telle que $M = \langle In, Out, Q, \mathsf{out}, E, q_0\rangle$. Considérons une autre machine $M' = \langle In', Out', Q', \mathsf{out}', E', q'_0\rangle$. Un état $q' \in Q'$* satisfait la spécification fonctionnelle *donnée par un état $q \in Q$ si et seulement si $q' \sqsubseteq_{g_{In}, g_{Out}} q$. Par extension, M' satisfait M si et seulement si il existe une relation de simulation R telle que $(q'_0, q_0) \in R$, c'est-à-dire ssi $q'_0 \sqsubseteq_{g_{In}, g_{Out}} q_0$. Afin que notre traitement reste générique, cela sera noté* $\mathsf{FuncSat}(q'_0, q_0)$ *et* $\mathsf{FuncSat}(M', M)$.

La structure de monoïde de nos alphabets nous permet différentes approches quant à l'interprétation des contraintes de délai de séparabilité. Comme montré en Sec. 3.3, il est possible de considérer les séparateurs valides sur au moins un contexte (quantification existentielle) ou sur tous les contextes (quantification universelle). Nous allons étudier la satisfaction des délais de séparabilité optimiste et pessimiste dans ces deux approches.

3.4.2.1 Satisfaction du délai de séparabilité

Nous nous plaçons dans les hypothèses de la Def. 33, munis d'une spécification $\langle M, spec \rangle$ et d'une implémentation M'. Il existe donc une relation de simulation $R \subseteq Q' \times Q$ entre les états de M' et ceux de M. La relation de satisfaction temporelle entre l'implémentation M' et la spécification $spec$ est définie ci-dessous.

Définition 34 (Relation de satisfaction temporelle)**.** *Le fait qu'une implémentation satisfasse une spécification temporelle est noté $M' \vDash spec$. Cette relation est classiquement définie par induction sur la formule $spec$. Nous pouvons considérer une notion de satisfaction optimiste et une notion de satisfaction pessimiste, en lien avec les différentes notions de délai de séparabilité étudiées dans ce manuscrit.*

$$
\begin{array}{lcll}
M' & \vDash & \phi_0 \wedge \phi_1 & \iff M' \vDash \phi_0 \textit{ et } M' \vDash \phi_1 \\
M' & \vDash & \phi_0 \vee \phi_1 & \iff M' \vDash \phi_0 \textit{ ou } M' \vDash \phi_1 \\
M' & \vDash & \langle I, O, t, q \rangle & \iff \forall q' \in R^{-1}(q), \mathsf{septime}^{\alpha}_{\mathsf{O}}(\prod[I,O](q')) \leq t \textit{ (cas optimiste)} \\
M' & \vDash & \langle I, O, t, q \rangle & \iff \forall q' \in R^{-1}(q), \mathsf{septime}^{\alpha}_{\mathsf{P}}(\tilde{\prod}[I,O](q')) \leq t \textit{ (cas pessimiste)}
\end{array}
$$

3.4.3 Prouver le délai de séparabilité par raffinement

Afin de tirer parti des résultats sur l'abstraction de machines (cf. Sec. 3.2), nous nous plaçons dans le cadre de la relation de simulation abstraite. Nous choisissons la relation de simulation inverse comme relation de raffinement. Posons l'existence de deux machines M et M' telles que décrites en Def. 33. .

Définition 35 (Raffinement)**.** *Soit $q \in Q$ un état de M et $q' \in Q'$ un état de M'. Nous dirons que q' raffine q si et seulement si $q' \sqsubseteq_{g_{In}, g_{Out}} q$.*

Afin de pouvoir prouver des propriétés incrémentalement, notre méthodologie doit apporter au développeur la garantie que les propriétés prouvées à certaines étapes de la conception sont *conservées* par l'opération de raffinement. Dans cette section, nous montrerons que la relation de raffinement (ici, la simulation abstraite inverse) conserve le délai de séparabilité, sous certaines hypothèses. La conservation des propriétés fonctionnelles ne sera pas traitée.

Nous nous plaçons dans le cadre posé par la définition du raffinement (Def. 35), c'est-à-dire en supposant l'existence de deux machines M et M' liées par deux correspondances de Galois, comme édicté plus haut.

3.4.3.1 Cas du délai de séparabilité optimiste

Nous avons prouvé dans le cas concret (Sec. 2.4.2 p. 42) que les séparateurs non-effectifs ne sont en général pas conservés par simulation inverse. Ce résultat s'étend au délai de séparabilité optimiste dans les cas concrets et abstraits. Le fait de raffiner un système ne garantit donc pas en général la conservation du délai de séparabilité optimiste des états (et donc par extension la relation de satisfaction n'est pas conservée).

Cependant, nous avons prouvé qu'il existe un ensemble de mots garantissant l'occurrence d'un effet observable et conservés par simulation inverse : les pseudo-séparateurs (cf. Sec. 2.4.2.4 p. 46). Rappelons que dans le cas concret, le délai de séparabilité d'un état est préservé

si et seulement si l'état en question est doté d'une paire séparante stable. Le délai de séparabilité peut alors être borné :

$$\forall q, \forall p \precsim q, \mathsf{septime}_{\mathsf{O}}(p) \leq \mathsf{max}\ \{\mathsf{septime}_{\mathsf{O}}(a_1, a_2) \mid (a_1, a_2) \in \mathsf{SepPairs}_*(q)\}.$$

Il nous reste à étendre ce résultat aux machines abstraites. Nous commençons par montrer l'existence d'un majorant permettant de sur-approximer le délai de séparabilité optimiste d'un état et de tous ses raffinements possibles.

Proposition 3.28 (Majoration du délai de séparabilité optimiste abstrait)**.** *Soit $q \in Q$ un état d'une machine M telle que définie plus haut. Définissons le majorant optimiste Opt :*

$$Opt(q) = \mathsf{max} \left\{ |w_*| \;\middle|\; \begin{array}{l} \exists (a_1, a_2) \in \mathsf{SepPairs}^{\alpha}_*(q), \exists b_1 \preccurlyeq a_1, b_2 \preccurlyeq a_2, \exists q_1, q_2 \in Q, \\ q \xrightarrow{b_1} q_1 \in E \wedge q \xrightarrow{b_2} q_2 \in E \implies w_* \in \mathsf{Minimal}(\mathsf{S}^{\alpha}_*(q_1, q_2)) \end{array} \right\}$$

Pour tout état $p \in Q'$ de M', la propriété suivante est vérifiée :

$$p \sqsubseteq_{g_{In}, g_{Out}} q \implies \mathsf{septime}^{\alpha}_{\mathsf{O}}(p) \leq Opt(q).$$

Démonstration. Considérons le triplet (q_1, q_2, w_*) tel que $|w_*|$ soit le pseudo-séparateur de longueur maximale pour q. Étudions maintenant un état p tel que $p \sqsubseteq q$. Les paires séparantes stables sont conservées par simulation inverse, nous pouvons donc affirmer l'existence de deux transitions $p \xrightarrow{c_1} p_1$ et $p \xrightarrow{c_2} p_2$ telles que $c_1 \preccurlyeq a_1 \ \wedge \ c_2 \preccurlyeq a_2$. Par simulation abstraite, il existe deux transitions $q \xrightarrow{d_1} q'_1$ et $q \xrightarrow{d_2} q'_2$ avec $p_1 \sqsubseteq q'_1$, $p_2 \sqsubseteq q'_2$ et $d_1 \preccurlyeq a_1 \wedge d_2 \preccurlyeq a_2$. Par la maximalité de $|w_*|$, nous avons :

$$\forall w \in \mathsf{Minimal}(\mathsf{S}^{\alpha}_*(q_1, q_2)), \mathsf{septime}^{\alpha}_{\mathsf{O}}(q'_1, q'_2) \leq |w| \leq |w_*|.$$

Afin de conclure, nous avons besoin de prouver que si $w \in \mathsf{S}^{\alpha}_*(q'_1, q'_2)$, alors il existe un mot $w' \in prefix(w)$ tel que $w' \in \mathsf{S}^{\alpha}_*(p_1, p_2)$. Comme montré dans le cas concret (Sec. 2.4.2.2 p. 44), cette propriété caractérise les pseudo-séparateurs (la preuve de cette caractérisation dans le cas abstrait est identique à celle du cas concret). □

Ceci motive une restriction de la relation de satisfaction, afin de garantir sa conservation par raffinement dans le cas optimiste.

Définition 36 (Relation de satisfaction temporelle stable, cas optimiste)**.** *Plaçons nous dans les hypothèses de la Def. 33 p. 85, munis d'une spécification fonctionnelle M et d'une implémentation M' telles que* $\mathsf{FuncSat}(M', M)$*. La relation de satisfaction temporelle stable est définie comme suit, pour le cas optimiste :*

$$M' \models^* \langle I, O, t, q \rangle \iff \left(\forall q' \in Q', q' \sqsubseteq_{g_{In}, g_{Out}} q \implies Opt(\textstyle\prod [I, O](q')) \leq t \right).$$

3.4.3.2 Cas du délai de séparabilité pessimiste

Contrairement au cas optimiste, le délai de séparabilité pessimiste de deux états est conservé par raffinement. Toutefois, cette propriété est perdue lorsque nous considérons le délai de séparabilité d'un unique état, c'est-à-dire le délai de séparabilité calculé en fonction des paires séparantes. En effet, de nouvelles paires séparantes peuvent naître dans les états raffinés. Afin

d'illustrer cela, il suffit d'observer l'exemple montrant la non-conservation du délai de séparabilité pessimiste par effacement (Fig. 3.9 p. 83) : nous avons montré que l'effacement est une abstraction ; l'opération réciproque est donc une simulation inverse. L'exemple dans la figure susmentionnée montre bien l'apparition de nouvelles paires séparantes.

Afin de pouvoir garantir la conservation du délai de séparabilité pessimiste, nous ne devons plus quantifier sur toutes les paires séparantes à chaque étape de raffinement, mais plutôt *fixer initialement* un ensemble de paires séparantes pour lesquelles nous pourront prouver un résultat de préservation. Des solutions arbitrairement plus fines peuvent être imaginées, mais l'idée centrale reste de paramétrer la relation de satisfaction temporelle par un ensemble de paires séparantes tel que nous puissions continuer à garantir le majorant temporel initial. Nous commençons par définir le délai de séparabilité paramétré par un ensemble de paires séparantes.

Définition 37 (Délai de séparabilité pessimiste paramétré)**.** *Soit $M = \langle In, Out, Q, \mathsf{out}, E, q_0\rangle$ une machine sur treillis et $q \in Q$ un état de M. Soit $F \subseteq \mathsf{SepPairs}^{\alpha}(q)$ un sous-ensemble des paires séparantes. Le délai de séparabilité pessimiste sur F est calculé en restreignant les états séparables à ceux atteignables par les paires de F :*

$$\mathsf{septime}^{\alpha}_{\mathsf{P}_F}(q) = \mathsf{max}\left\{ \mathsf{septime}^{\alpha}_{\mathsf{P}}(q_1,q_2) \;\middle|\; \begin{array}{l} \exists (a_1,a_2)\in F, \exists q_1,q_2\in Q, \exists b_1 \preccurlyeq a_1, b_2 \preccurlyeq a_2, \\ q \xrightarrow{b_1} q_1 \in E \wedge q \xrightarrow{b_2} q_2 \in E \end{array} \right\}.$$

Une propriété dont la démonstration est directe est que la restriction à un sous-ensemble des séparateurs sous-approxime le délai de séparabilité pessimiste. Il convient donc de justifier cette méthode en la replaçant dans le cadre du développement de systèmes temps-réels embarqués. En particulier, nous devons justifier du fait que cette méthode ne compromet pas la sûreté du système. Se restreindre à un ensemble fixé de paires séparantes revient à désigner à l'avance les données particulières sur lesquelles le système doit être réactif. Il se peut que d'autres données provoquent une réaction du système en un temps ne respectant pas la spécification, mais cela ne peut poser de problème que si *le reste du système* se repose sur les effets observables produits par ces données "non contrôlées" pour constituer des chemins de calcul critiques.

En d'autres termes, nous pouvons ignorer certaines paires séparantes durant le développement par raffinement si nous empêchons les effets observables générés par ces paires séparantes d'être pris en compte par le reste du système. Ces pré-conditions à la correction de la démarche par raffinement posées, nous pouvons prouver la conservation du délai de séparabilité pessimiste restreint à un sous-ensemble des paires séparantes.

Proposition 3.29. *Soit $F \subseteq \mathsf{SepPairs}^{\alpha}_{*}(q)$ un ensemble de paires séparantes stables. Soit $q' \in Q'$ un état quelconque de M'. La propriété suivante est vérifiée :*

$$p \sqsubseteq q \implies \mathsf{septime}^{\alpha}_{\mathsf{P}_F}(p) \leq \mathsf{septime}^{\alpha}_{\mathsf{P}_F}(q).$$

Démonstration. Soit $\mathsf{septime}^{\alpha}_{\mathsf{P}_F}(q)$ tel que défini en Def. 37.

- Si $\mathsf{septime}^{\alpha}_{\mathsf{P}_F}(q) = \infty$, le but suit immédiatement.
- Sinon, considérons la paire séparante (aq_1, aq_2) permettant d'atteindre le couple (q_1, q_2) d'états tel que $\mathsf{septime}^{\alpha}_{\mathsf{P}}(q_1,q_2)$ soit maximal (et dans ce cas, fini). Nous avons donc la propriété $q_1 \;)\!(^{\alpha}\; q_2$. Par définition, nous avons également :

$$\mathsf{septime}^{\alpha}_{\mathsf{P}}(q_1,q_2) = \mathsf{max}\,\{|w| \mid \forall w' \in \mathsf{prefix}(w), w' \notin \mathsf{S}^{\alpha}(p,q)\}.$$

Par la contrainte $q_1 \ )\!(^{\alpha}\ q_2$, le mot de longueur maximale induisant le temps $\mathsf{septime}^{\alpha}_{\mathsf{P}}(q_1, q_2)$ est également un pseudo-séparateur. Soit w_q ce pseudo-séparateur de longueur maximale pour (q_1, q_2).
Considérons maintenant p tel que $p \sqsubseteq q$ et considérons la paire séparante (ap_1, ap_2) permettant d'atteindre les états $p_1, p_2 \in Q'$ induisant $\mathsf{septime}^{\alpha}_{\mathsf{P}\,F}(p_1, p_2)$ maximal, ainsi que w_p le pseudo-séparateur de longueur maximale pour (p_1, p_2). Par définition de la simulation, il existe $a_1 \preccurlyeq ap_1$ et $a_2 \preccurlyeq ap_2$ ainsi que deux transitions $q \xrightarrow{a_1} q'_1 \in E$ et $q \xrightarrow{a_2} q'_2 \in E$ telles que $p_1 \sqsubseteq q'_1$ et $p_2 \sqsubseteq q'_2$. Enfin, soit $w_{q'}$ le pseudo-séparateur de longueur maximale pour (q'_1, q'_2). La caractérisation inductive des pseudo-séparateurs (Sec. 2.4.2.2 p. 44) nous permet d'affirmer que $|w_p| \leq |w_{q'}|$. Or, par maximalité de w_q, nous avons $|w_{q'}| \leq |w_p|$. Nous pouvons donc conclure par transitivité que $|w_p| \leq |w_q|$, ce qui achève de prouver la proposition.

□

Nous proposons également une notion de satisfaction temporelle stable pour le cas pessimiste. Contrairement au cas optimiste, cette relation de satisfaction n'*implique pas* la satisfaction temporelle dans le cas général.

Définition 38 (Relation de satisfaction temporelle stable, cas pessimiste)**.** *Soit M une spécification fonctionnelle et M' une implémentation, telles que* $\mathsf{FuncSat}(M',\ M)$*. Soit $F \subseteq \mathcal{T}_{In} \times \mathcal{T}_{In}$ un ensemble de paires d'entrées. La relation de satisfaction temporelle* stable*, pour le délai de séparabilité pessimiste, est définie comme suit.*

$$(M', F) \models^{*} \langle I, O, t, q\rangle \iff \forall q' \in R^{-1}(q), \mathsf{septime}^{\alpha}_{\mathsf{P}\,F}(\tilde{\prod}[I, O](q')) \leq t.$$

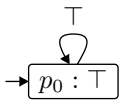
Afin d'obtenir une relation de satisfaction qui soit utilisable dans le cadre d'un formalisme de développement par raffinement, nous devons donc procéder en sélectionnant un ensemble particulier de paires séparantes F *à priori*, ce qui en pratique peut se révéler dur, tant sur le plan méthodologique que calculatoire.

3.4.4 Exemple

Nous illustrons le processus de raffinement par l'implémentation progressive d'une machine calculant un délai de trois unités de temps (rappelons que par essence, une machine de Moore induit un délai d'au moins une unité de temps). Nous présenterons en parallèle aux machines successivement raffinées, les programmes correspondant qui seront donnés dans un langage inspiré de PsyC, le langage de programmation de la suite OASIS [32]. Ceci nous permettra d'illustrer l'intérêt de contrôler les propriétés de la machine à états sous-jacente au texte du programme. Les portions de programmes indéfinies seront notées par •. La syntaxe et la sémantique opérationnelle de ce langage sont définies en Annexe A. Nous imposons que le résultat doit être une machine appartenant à $\mathsf{Moore}(\mathbf{bool}, \mathbf{bool})$. Noter que nous n'utilisons pas les résultats sur l'extension du délai de séparabilité au cadre multi-ports. Enfin, nous considérerons le délai de séparabilité dans sa variante optimiste.

Spécification. Nous ne spécifions aucun comportement fonctionnel particulier. La spécification temporelle se traduit en une contrainte de délai de séparabilité de 3 unités de temps sur tous les états de la machine, entre l'unique port d'entrée et l'unique port de sortie (nous omettrons

les annotations de ports, qui sont superflues ici). Nous partons d'une spécification $\langle M_i, \langle 3, q_0 \rangle\rangle$ où M_i est la machine suivante, qui est la plus générale possible étant donnée la spécification.



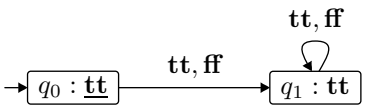
Cette machine correspond à tous les programmes possibles dont l'exécution se poursuit indéfiniment.

Première étape. Nous fixons la valeur observable initiale de la machine à **tt** , arbitrairement.

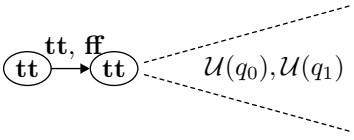
advance$(1,$ **tt** $)$;
while tt do
  advance$(1, \bullet)$;
done

Deuxième étape. Nous implémentons la mémoire tampon par deux variables x_0 et x_1.

x_0 := **tt** ;
x_1 := **tt** ;
advance$(1,$ **tt** $)$;
while tt do
  advance$(1, !x_0)$;
  x_1 :=$!x_0$;
  x_1 := **get**$_0$
done

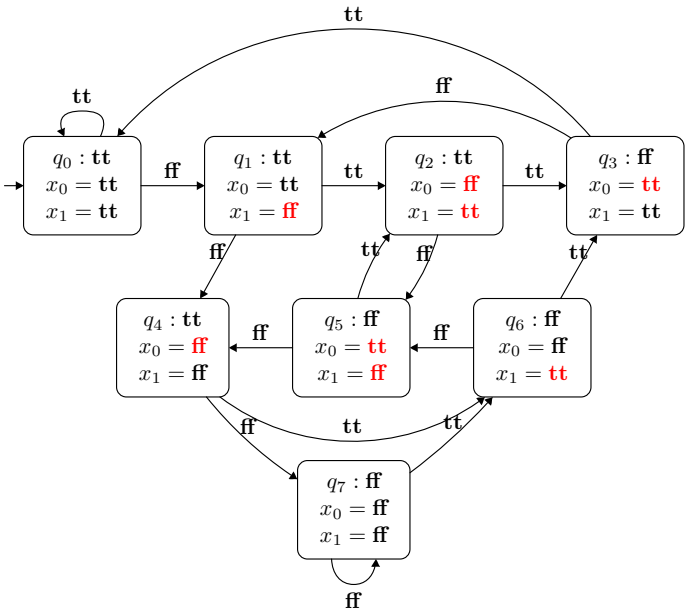


Il est direct de montrer que $q_0 \sqsubseteq p_0$ et $q_1 \sqsubseteq p_0$, ce qui suffit à montrer la satisfaction fonctionnelle. Afin de prouver la satisfaction temporelle, nous devons montrer pour q_0 et q_1 l'existence d'un paire séparante pour laquelle le délai de séparabilité est inférieur ou égal à 3. Etudions le dépliage de q_0.



Ce dépliage montre clairement que $\mathsf{SepPairs}^{\alpha}(q_0) = \varnothing$, ce qui implique $\mathsf{septime}^{\alpha}_{\mathsf{O}}(q_0) = \infty$. Qui plus est, le programme est déterministe, ce qui induit l'impossibilité de raffiner plus avant. La raison est que nous avons fait une erreur lors du raffinement du programme source en intervertissant les variables x_0 et x_1 lors de la mise à jour de la mémoire tampon. La version correcte est donnée ci-dessous.

x_0 := **tt** ;
x_1 := **tt** ;
advance(1, **tt**);
while tt do
  advance(1, $!x_0$);
  x_0 :=$!x_1$;
  x_1 := **get**$_0$
done



La correction de ce dernier raffinement est montrée en chaque état par l'existence de la paire séparante (**tt**, **ff**) et de l'ensemble de séparateurs **bool**^{2} (en d'autres termes,tous les mots de longueur 2 sont des séparateurs). La nature déterministe du programme résultant permet de vérifier ce fait par simple examen des traces de sortie.

3.5 Synthèse

Ce chapitre a exposé l'utilisation de la relation de simulation afin de mieux concevoir et analyser les systèmes synchrones.

Nous avons introduit une notion de machine abstraite faisant usage de données tirées de treillis, afin de spécifier à haut niveau certaines contraintes fonctionnelles simples. Les notions de délai de séparabilité ont été étendues à ce cadre et des résultats de conservation permettent de garantir la correction du système final vis-à-vis de la spécification.

La notion d'abstraction a également été utilisée afin d'étudier comment analyser en pratique le délai de séparabilité entre des ports d'entrée et de sortie particuliers. Nous avons montré que le fait d'effacer les données sur les ports non considérés est une forme particulière d'abstraction et nous avons défini des conditions suffisantes à l'analysabilité du délai de séparabilité par cette méthode.

Enfin, nous avons appliqué la notion de simulation inverse au développement par raffinement de machines de Moore. Nous avons à nouveau montré qu'une restriction du délai de séparabilité est préservée par simulation, ce qui entraîne la correction *par construction* des systèmes concrets résultants.

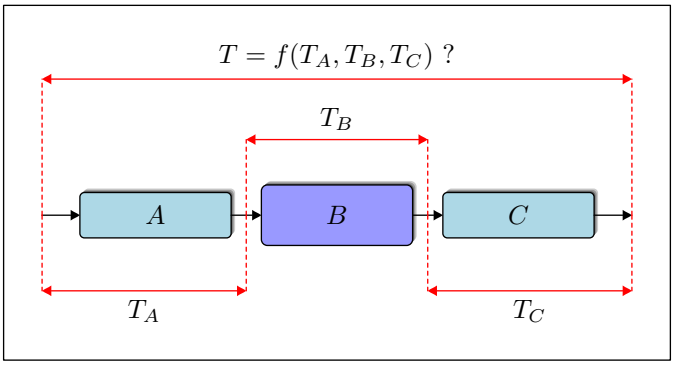
4

Composition et délai de séparabilité

Nous avons étudié au chapitre précédent comment construire des systèmes incrémentalement par raffinement, tout en garantissant la conservation du délai de séparabilité. Ce chapitre propose de compléter cette démarche par une étude de la conservation de certaines propriétés de séparabilité lors de la *composition* de systèmes. L'objectif est de permettre de raisonner de façon *modulaire* sur le délai de séparabilité de certaines parties du système.

Nous commençons par définir quelques opérations de composition de machines : la composition séquentielle, la composition parallèle et la boucle de rétroaction. Nous procédons alors à une étude détaillée du comportement des effets observables lors de l'opération de composition séquentielle. Les résultats négatifs de cette étude nous mènent à proposer une méthode approchée permettant de recouvrer un comportement modulaire au prix d'une certaine perte d'information. Pour plus de simplicité, nous travaillerons dans ce chapitre sur les machines de Moore concrètes.

Le type de situation que nous voulons étudier est illustré par la figure suivante.



Etant donné un ensemble de sous-systèmes A, B et C tous dotés de leurs délais de séparabilité (ici, respectivement T_A, T_B et T_C), nous voulons déterminer un moyen simple de calculer T. Dans notre contexte, cet objectif de simplicité peut prendre plusieurs significations :

- le calcul de T ne doit idéalement reposer que sur les délais T_A, T_B, T_C,
- la complexité doit croître linéairement avec la taille du système lors du développement.

4.1 Algèbre de processus basée sur les machines de Moore

Nous construisons une algèbre de processus autour des machines de Moore. Cette algèbre est définie par un ensemble d'opérateurs de composition permettant de structurer des ensembles de machines de Moore communicantes. Nous nous distinguons en particulier d'Argos [51, 52]

par le fait que ce dernier est basé sur les machines de Mealy et repose donc sur le modèle synchrone fort. De plus, Argos permet la conception hiérarchique de systèmes synchrones, ce que nous ne proposons pas.

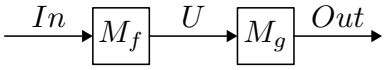
Nous avons opté pour une définition minimaliste inspirée des travaux d'Abramsky sur l'étude des algèbres de processus dans le cadre de la théorie des catégories [2]. Nous travaillerons dans un cadre ensembliste plus classique, mais cette affiliation justifie nos trois opérations de composition : composition parallèle non-communicante, composition séquentielle et boucle de rétroaction [2].

4.1.1 Composition séquentielle de machines

Soient M_f et M_g deux machines :

$$\begin{array}{rcl} M_f & = & \langle In, U, Q_f, E_f, \mathsf{out}_f, q_{i,f} \rangle, \\ M_g & = & \langle U, Out, Q_g, E_g, \mathsf{out}_g, q_{i,g} \rangle. \end{array}$$

La composition séquentielle de M_f et M_g est intuitivement la "mise en série" des machines M_f et M_g. Il s'agit d'une composition parallèle communicante, où la communication s'effectue de manière non-blocante de M_f vers M_g. Intuitivement, M_f traite le flot d'entrée, puis le résultat est transmis à M_g. Graphiquement, cette composition peut être décrite ainsi :



Définition 39 (Composition séquentielle)**.** *La composition séquentielle $M_g \circ M_f$ procède en redirigeant la sortie de M_f à l'entrée de M_g. La machine composée est :*

$$M_g \circ M_f = \langle In, Out, Q_{g \circ f}, E_{g \circ f}, \mathsf{out}_{g \circ f}, (q_{i,f}, q_{i,g}) \rangle,$$

dont les composantes sont définies ci-dessous :

$$\begin{array}{lcl} Q_{g \circ f} & = & Q_f \times Q_g \\ E_{g \circ f} & = & \{(q_f, q_g) \xrightarrow{in_f} (q'_f, q'_g) \mid q_f \xrightarrow{in_f} q'_f \in E_f \ \wedge \ q_g \xrightarrow{\mathsf{out}(f)} q'_g \in E_g\} \\ \mathsf{out}_{g \circ f}(q_f, q_g) & = & \mathsf{out}_g(q_g) \end{array}$$

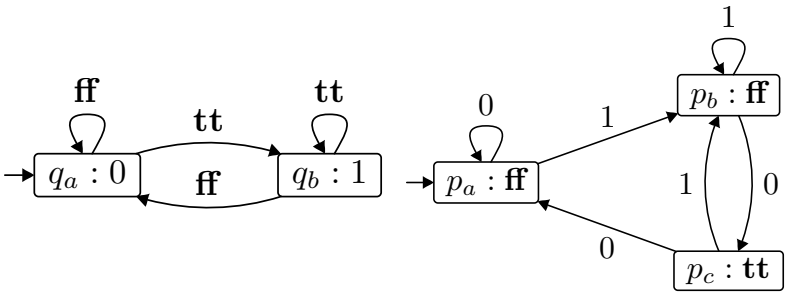
Exemple 4.1 (Composition séquentielle) **:** La figure 4.1 présente un exemple de composition séquentielle. La machine M_f (déjà présentée plus haut en Fig. 4.1(a)) associe l'entier 0 au booléen **ff** et l'entier 1 au booléen **tt**. La machine M_g (Fig. 4.1(b)) est un détecteur de front descendant. Leur composition est présentée en Fig. 4.1(c).

Une propriété désirable de la composition séquentielle est l'associativité.

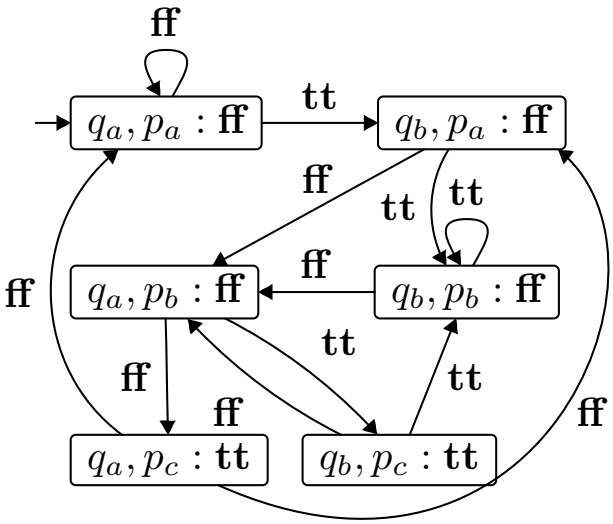
Lemme 4.2 (Associativité de la composition séquentielle)**.** *Pour tout $A, B, C, D \in \mathfrak{D}$ et pour toutes machines $M_f \in \mathsf{Moore}(A, B), M_g : \mathsf{Moore}(B, C), M_h : \mathsf{Moore}(C, D)$, l'équivalence suivante est vérifiée :*

$$M_h \circ (M_g \circ M_f) \simeq (M_h \circ M_g) \circ M_f.$$

Démonstration. La preuve est directe, par bisimulation. □



(a) Machine M_f (b) Machine M_g



(c) Machine $M_{g \circ f}$

FIGURE 4.1 – Exemple de composition séquentielle

4.1.2 Composition parallèle de machines

L'opération de composition parallèle que nous utilisons n'induit pas de communication : elle correspond à leur simple "juxtaposition" synchrone. Le choix inverse est possible, comme montré par exemple dans les travaux de Clarke et al. [28].

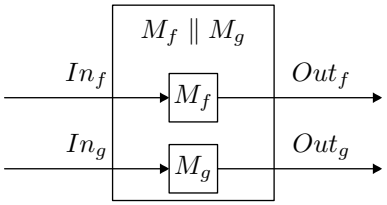
Nous avons vu dans la définition de $\mathfrak{D}$ (Sec. 2.1.2 p. 24) que nos types de données sont des produits cartésiens d'ensembles finis. Cette structure peut être étendue aux ensembles de machines par homomorphisme et c'est l'opération de ce monoïde qui fera office de composition parallèle.

Définition 40 (Composition parallèle)**.** *Soient In_f, Out_f, In_g, Out_g quatre ensembles. Soient $M_f = \langle In_f, Out_f, Q_f, E_f, \mathsf{out}_f, q_{i,f}\rangle$ et $M_g = \langle In_g, Out_g, Q_g, E_g, \mathsf{out}_g, q_{i,g}\rangle$ deux machines. La composition parallèle procède en appariant les transitions respectives de M_f et M_g de façon synchrone. La machine composée est :*

$$M_f \parallel M_g = \langle In_f \times In_g, Out_f \times Out_g, Q_{f\parallel g}, E_{f\parallel g}, \mathsf{out}_{f\parallel g}, (q_{i,f}, q_{i,g})\rangle.$$

$$\begin{array}{lcl} Q_{f\parallel g} & = & Q_f \times Q_g \\ E_{f\parallel g} & = & \{(q_f, q_g) \xrightarrow{\langle in_f, in_g\rangle} (q'_f, q'_g) \mid q_f \xrightarrow{in_f} q'_f \in E_f \ \wedge \ q_g \xrightarrow{in_g} q'_g \in E_g\} \\ \mathsf{out}_{f\parallel g}(q_f, q_g) & = & (\mathsf{out}_f(q_f), \mathsf{out}_g(q_g)). \end{array}$$

L'opération de composition parallèle est illustrée par le schéma suivant :



Exemple 4.3 (Composition parallèle) **:** La figure 4.2 montre le résultat de la composition parallèle des machines M_f (en Fig. 4.1(a)) et M_g (en Fig. 4.1(b)). Le résultat a un nombre d'états et de transitions égal au produit de ses composantes. On reconnaît dans la figure en question la machine M_f dupliquée à chaque "étage" (un étage par état de M_g).

L'associativité du produit de $\mathfrak{D}$ induit celle de la composition parallèle.

Proposition 4.4 (Associativité de $\parallel$)**.** *Pour tous $A, B, C, D, E, F \in \mathfrak{D}$ et pour toutes machines $M_f \in \mathsf{Moore}(A, B)$, $M_g \in \mathsf{Moore}(C, D)$, $M_h \in \mathsf{Moore}(E, F)$,*

$$M_f \parallel (M_g \parallel M_h) \simeq (M_f \parallel M_g) \parallel M_h.$$

Démonstration. La preuve procède de façon directe, par bisimulation, en utilisant l'associativité de $\mathfrak{D}$ directement sur les données en sortie et sur les arcs. □

La classe d'équivalence pour la bisimilarité des éléments de $\mathsf{Moore}(\mathbb{1}, \mathbb{1})$ fait office d'élément neutre pour la composition parallèle. Nous pouvons conclure de l'associativité de $\parallel$ et de l'existence d'un élément neutre que l'ensemble des machines est un monoïde pour la composition parallèle.

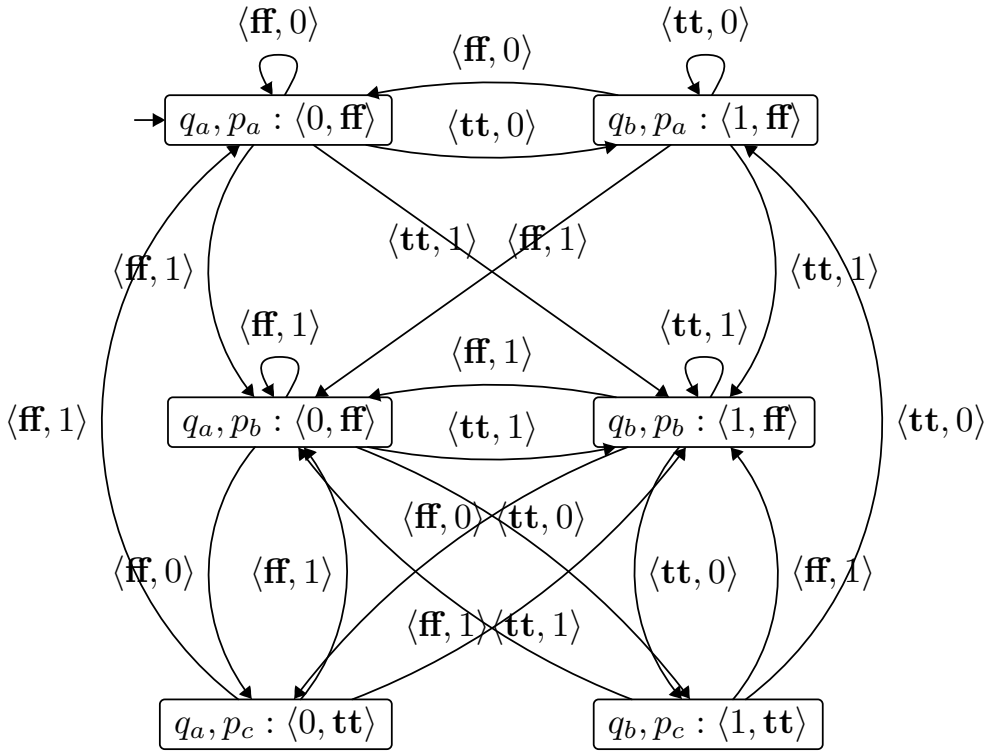


FIGURE 4.2 – Exemple de composition parallèle : $M_{f \| g}$

4.1.3 Boucle de rétroaction

Si nous nous limitions aux opérations de composition séquentielle et parallèle, le système final résultant de telles compositions aurait une structure de graphe acyclique dirigé. L'observation de systèmes existants montre la nécessité de pouvoir définir des *boucles* de rétroaction, permettant à un sous-programme de renvoyer des données en amont dans le flot de calcul (un exemple simple est un intégrateur numérique). C'est pourquoi nous définissons une opération de boucle de rétroaction, notée feedback. Cette opération procède en redirigeant à chaque transition les sorties du temps t aux entrées du temps $t + 1$ sur une paire de ports donnée de même type.

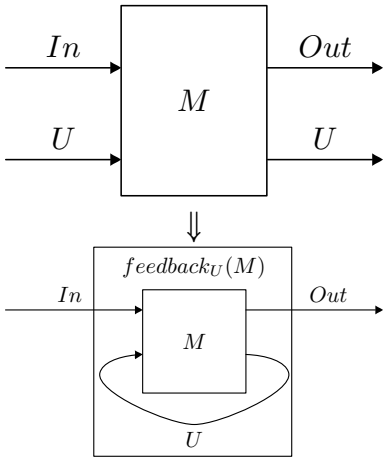
Définition 41 (Boucle de rétroaction)**.** *Soient $In, Out, U \in \mathfrak{D}$ et soit une machine :*

$$M = \langle In \times U, Out \times U, Q, E, \mathsf{out}, q_i \rangle.$$

L'opération permettant de connecter la sortie sur U à l'entrée sur U est la boucle de rétroaction $\mathsf{feedback}_U(M) = \langle In, Out, Q_{fb}, E_{fb}, \mathsf{out}_{fb}, q_i \in E \rangle$*, dont les composantes sont définies ci-dessous :*

$$\begin{array}{lcl} Q_{fb} & = & Q \\ E_{fb} & = & \{q \xrightarrow{a} q' \mid q \xrightarrow{\langle a, \pi_U(\mathsf{out}(q)) \rangle} q'\} \\ \mathsf{out}_{fb}(q) & = & \pi_{In}(\mathsf{out}(q)). \end{array}$$

Cette opération de composition est représentée dans la figure suivante.



Exemple 4.5 : Il est intéressant de voir que la composition séquentielle peut être exprimée par l'utilisation de la composition parallèle et de la boucle de rétroaction. Considérons les machines $M_f \in \mathsf{Moore}(\mathbf{bool}, \{0;1\})$ et $M_g \in \mathsf{Moore}(\{0;1\}, \mathbf{bool})$ montrées en Fig. 4.1. Leur composition parallèle, montrée en Fig. 4.2 p. 97, est $M_{f\|g} \in \mathsf{Moore}(\mathbf{bool} \times \{0;1\}, \{0;1\} \times \mathbf{bool})$. Modulo la commutativité de $\mathfrak{D}$, nous pouvons écrire $M_{f\|g} \in \mathsf{Moore}(\mathbf{bool} \times \{0;1\}, \mathbf{bool} \times \{0;1\})$. Alors, la machine $\mathsf{feedback}_{\mathbf{bool}}(M_{f\|g})$ calculée en redirigeant la sortie de booléens sur l'entrée est exactement la machine $M_{g\circ f}$ en Fig. 4.1(c).

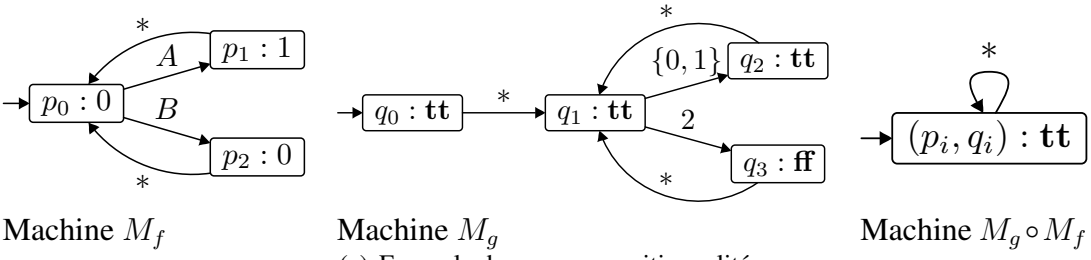
4.1.4 Notes sur la structure de l'ensemble des machines

Dans cette section, nous avons défini une algèbre de processus synchrones inspirée des travaux d'Abramsky [3]. Ce dernier propose une algèbre de processus dans un langage catégorique, où les morphismes sont des machines de Mealy et les objets des ensembles de mots particuliers. Cette catégorie est dotée d'une structure monoïdale donnée par la composition parallèle et la composition de morphismes est analogue à la composition séquentielle que nous avons défini plus haut. L'opérateur de feedback correspond à une structure additionnelle appelée *trace* [43]. Un théorème général lui permet d'affirmer que cet ensemble de primitives est, en un sens, *complet* : toutes les diagrammes constitués de machines connectées entre elles peuvent être exprimés par l'utilisation de ces primitives et de quelques isomorphismes structurels : commutativité, associativité, duplication de ports, effacement de ports, absorption de l'identité pour la composition parallèle. Il est important de noter que la construction d'Abramsky fonctionne parce que les machines de Mealy sont capables d'exprimer la fonction identité sur les traces (condition nécessaire à la bonne formation d'une catégorie), ce dont les machines de Moore sont incapables.

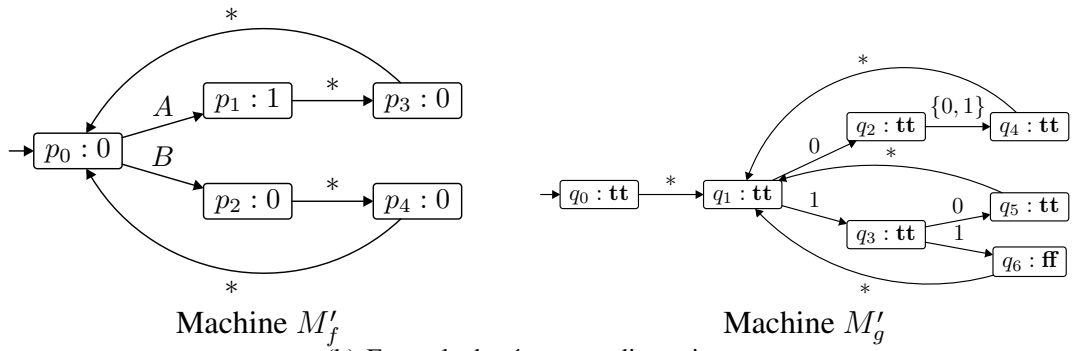
Notre ensemble de primitives n'est en ce sens pas complet, mais puisque ni les primitives structurelles évoquées plus haut ni la fonction identité n'ont d'impact sur le délai de séparabilité, nous n'avons pas cherché à enrichir notre algèbre de façon à la doter de la structure adéquate.

4.2 Effets observables sous composition séquentielle

Dans le premier chapitre, nous avons défini une notion d'effet observable pour les machines de Moore. Dans cette section, nous étudions le comportement des effets observables lors de la composition séquentielle de machines de Moore et ses conséquences sur le délai de séparabilité.



(a) Exemple de non-compositionnalité



(b) Exemple de séparateur disparaissant

FIGURE 4.3 – Contre-exemples à la conservation de la réactivité par composition séquentielle

Nous avons vu que la réactivité est une propriété des états (cf. Def. 6 p. 36). Lors de la composition séquentielle de deux machines, nous devrons donc considérer la réactivité de l'état résultant en fonction des états le composant. Afin d'illustrer la dynamique des effets observables, nous étudions quelques exemples où la propriété de réactivité est perdue. Ces exemples suivent le même raisonnement montrant que la composition de deux fonctions totales non-constantes peut être constante.

Plus précisément, ils illustrent deux conditions nécessaires afin que la propriété de réactivité soit conservée :

– un effet observable doit être "pris en compte" par une paire séparante d'un état récepteur ;
– la paire séparante réceptrice doit exister dans le contexte induit par la machine émettrice.

La figure 4.3(a) montre la composition de deux machines de Moore réactives M_f et M_g dont la composition n'est pas réactive. Dans ce premier exemple, une condition suffisante afin d'obtenir une machine composée réactive serait que l'effet observable $(0, 1)$ (induit par un séparateur de longueur 0) produit par p_0 dans M_f soit "pris en compte" par la machine M_g, c'est-à-dire que $(0, 1)$ soit une paire séparante de q_1. Ce n'est pas le cas et le résultat de la composition est une machine constante.

L'exemple montré en Fig. 4.3(b) illustre une situation plus complexe et montre que la condition, qui était suffisante sur l'exemple précédent, ne l'est pas en général. Dans M'_f et M'_g, les états p_0 et q_1 pris isolément sont réactifs. Cette propriété est perdue lorsque ils sont composés : les données en sortie des états p_3 et p_4 imposent à q_2 et q_3 un **contexte** réduit au mot 0. Dans ce contexte, le mot séparateur 1 de q_2 et q_3 n'existe pas et ces états ne sont plus distinguables l'un de l'autre. Ainsi q_2 et q_3 ne sont plus séparables et le résultat est la machine constante vue plus haut. Le fait qu'un effet observable de M'_f corresponde à une paire séparante de M'_g n'est donc pas suffisant pour garantir un effet observable en sortie. La composition séquentielle restreint le langage d'entrée du système en position réceptrice (ici, M'_g). Cela signifie que les séparateurs peuvent d'une manière générale disparaître et apparaître de façon arbitraire.

Ces contre-exemples confirment l'intuition : il n'existe pas de moyen général de garantir et de composer des dépendances fonctionnelles. Afin de pouvoir contourner cet obstacle, nous devons déterminer les conditions nécessaires et suffisantes pour qu'une composition d'états soit réactive. Ce problème se traduit ainsi : étant données deux machines $M_f \in \mathsf{Moore}(In, U)$ et $M_g \in \mathsf{Moore}(U, Out)$, et deux paires d'états composés $(q_f, q_g) \in Q_{g \circ f}$, $(p_f, p_g) \in Q_{g \circ f}$, quelles sont les conditions nécessaires et suffisantes pour que $(q_f, q_g) \not\sim (p_f, p_g)$?

Il apparaît de façon directe que si nous avons $q_f \sim p_f$ et $q_g \not\sim p_g$, alors nous obtiendrons $(q_f, q_g) \sim (p_f, p_g)$. Une condition nécessaire est donc que $q_f \not\sim p_f$ ou $q_g \not\sim p_g$, mais c'est pas une condition suffisante. Dans la suite, nous allons étudier en détail la structure d'un contre-exemple à la bisimilarité des états composés (q_f, q_g) et (p_f, p_g), et en particulier la façon dont ces contre-exemples sont *construits* à partir des contre-exemples à la bisimilarité de q_f et p_f d'une part, et de q_g et p_g d'autre part.

A cette fin, nous définisons la notion de *candidat de preuve de non-bisimilarité*, qui sont des arbres ayant le potentiel d'être des contres-exemples mais dont la structure est trop lâche. Nous étudierons sous quelles conditions ces candidats deviennent de véritables contres-exemples, et nous nous servirons de ces résultats pour étudier la construction d'une preuve de non-bisimilarité pour des états composés.

Dans la suite, nous nous plaçons dans le cadre de la composition séquentielle, comme définie en Def. 39 p. 94.

4.2.1 Notion de candidat de preuve de non-bisimilarité

Considérons une machine $M = \langle In, Out, Q, E, \mathsf{out}, q_0 \rangle$ et soient $p, q \in Q$ deux états quelconques. Dans le chapitre 2, nous avons proposé une caractérisation logique explicite des contre-exemples à la bisimilarité de deux états p et q (Prop. 2.7 p. 35). Ces contre-exemples, qui sont donc des *preuves* de la non-bisimilarité de p et q, sont définis inductivement. Chaque contre-exemple est un *arbre de dérivation* particulier construit à l'aide des trois règles BASE, SYM et IND. La structure exacte de chaque contre-exemple est définie en interprétant les prémisses de chaque règle dans le cadre d'une logique constructive. Cependant, le présent manuscrit n'a pas pour but d'effectuer une présentation détaillée de ces concepts et nous ne justifierons pas nos constructions autrement que par l'exemple. Le lecteur voulant plus de détails pourra par exemple se référer à [36].

Rappelons que la notation utilisée dans la suite :

$$\prod_{x \in A} B(x)$$

correspond à l'ensemble des fonctions de domaine A et dont le codomaine $B(x)$ varie selon l'argument x de la fonction en question. Dans les développements qui suivent, nous aurons en particulier besoin de fonctions associant à des transitions, des candidats de preuve non-bisimilarité. Nous définissons ici la structure générale d'un arbre candidat de preuve de non-bisimilarité, inductivement. Les sections qui suivent utiliseront ces arbres candidats pour construire des arbres *preuves* de la non-bisimilarité de deux états lors d'une composition séquentielle.

Dans la définition qui suit, noter que la seule réelle différence avec les contres-exemples définis au chapitre 2 (outre les détails de formalisation) est que les feuilles, constituées par le cas de base, ne produisent pas forcément des sorties différentes.

Définition 42 (Arbre candidat)**.** *Soient deux états $p \in Q$ et $q \in Q$. L'ensemble des arbres candidats pour p et q est noté* $\mathsf{Candidates}_M(p,q)$.

- ***(Cas de base.)*** *Nous avons* $\textsc{Base}(p,q) \in \mathsf{Candidates}_M(p,q)$.
- ***(Cas symétrique.)*** *Pour tout* $h \in \mathsf{Candidates}(q,p)$*, nous avons* $\textsc{Sym}(h) \in \mathsf{Candidates}_M(p,q)$.
- ***(Cas inductif.)*** *L'arbre de dérivation est construit selon la prémisse de la règle* Ind. *Soit* $p \xrightarrow{a} p' \in E$ *une transition et soit :*

$$s \in \prod_{q \xrightarrow{a} q'} \mathsf{Candidates}_M(p',q')$$

une fonction associant à toute transition $q \xrightarrow{a} q' \in E$ *un arbre candidat pour* p' *et* q'*. Nous avons alors :*

$$\textsc{Ind}(p \xrightarrow{a} p', s) \in \mathsf{Candidates}_M(p,q).$$

Puisque l'espace d'états et les alphabets sont finis, nous représentons explicitement chaque fonction s par son graphe. Nous pouvons noter que pour tout couple d'état (p,q), toute preuve de non-bisimilarité est un arbre candidat particulier, c'est-à-dire :

$$h \in \mathsf{c{-}ex}(p,q) \implies h \in \mathsf{Candidates}_M(p,q).$$

La réciproque n'est pas vraie, car le cas de base de construction des arbres candidats n'impose pas $\mathsf{out}(p) \neq \mathsf{out}(q)$.

Exemple 4.6 : La figure 4.4 illustre la construction d'une preuve de non-bisimilarité pour le cas inductif. Cette figure représente deux états composés (p_f, p_g) et (q_f, q_g) tels qu'il existe un arc particulier $(p_f, p_g) \xrightarrow{in_f} (p'_f, p'_g)$ ayant pour propriété la possibilité d'associer à tout arc $(q_f, q_g) \xrightarrow{in_f} (q^i_f, q^j_g)$ une preuve de non-bisimilarité de $\mathsf{c{-}ex}((p'_f, q'_f), (q^i_f, q^j_g))$. En termes d'arbres candidats, cette figure correspond à l'élément :

$$\textsc{Ind}((p_f, p_g) \xrightarrow{in_f} (p'_f, p'_g), s);$$

où s est précisément la fonction associant à tout arc $(q_f, q_g) \xrightarrow{in_f} (q^i_f, q^j_g)$ un contre-exemple appartenant à $\mathsf{c{-}ex}((p'_f, q'_f), (q^i_f, q^j_g))$.

Feuilles d'un arbre candidat. Par définition, l'ensemble des feuilles d'un arbre candidat est l'ensemble des couples d'états $\textsc{Base}(p,q)$ atteignables depuis la racine.

Définition 43 (Feuilles d'un arbre candidat)**.** *Nous nous plaçons dans le contexte de la Def. 42 sur les arbres candidats. Si $p \in Q$ et $q \in Q$ sont deux états et $h \in \mathsf{Candidates}_M(p,q)$ est un arbre candidat, l'ensemble des feuilles de h est noté* $\mathsf{leaves}(h)$ *et est défini récursivement sur h.*

$$\begin{aligned}
\mathsf{leaves}(\textsc{Base}(p,q)) &= \{(p,q)\} \\
\mathsf{leaves}(\textsc{Sym}(h)) &= \mathsf{leaves}(h) \\
\mathsf{leaves}(\textsc{Ind}(p \xrightarrow{a} p', s)) &= \bigcup_{q \xrightarrow{a} q' \in E} \mathsf{leaves}\Big(s(q \xrightarrow{a} q')\Big)
\end{aligned}$$

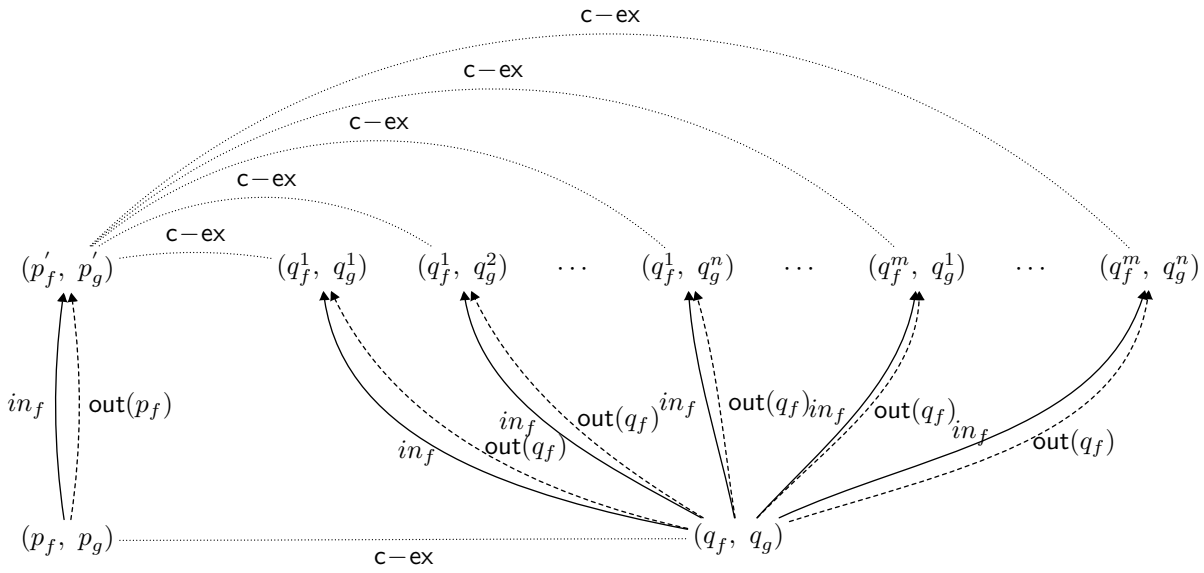


FIGURE 4.4 – Illustration de la construction d'une preuve de non-bisimilarité, cas inductif

Conditions sous lesquelles un arbre candidat est une preuve de non-bisimilarité.
Le lemme suivant permet d'établir les conditions pour qu'un arbre candidat soit une preuve de non-bisimilarité.

Lemme 4.7. *Nous nous plaçons à nouveau dans le contexte de la Def. 42 sur les arbres candidats. Soient $p \in Q$ et $q \in Q$ deux états. Pour tout arbre candidat $h \in \mathsf{Candidates}_M(p,q)$ quelconque, nous avons :*

$$(\forall (p',q') \in \mathsf{leaves}(h), \exists h' \in \mathsf{c{-}ex}(p',q')) \implies \mathsf{c{-}ex}(p,q) \neq \varnothing.$$

En d'autres termes, h est une preuve de non-bisimilarité si toutes les feuilles de h sont telles que les états correspondants soient non-bisimilaires.

Démonstration. La preuve suit par définition de $\mathsf{c{-}ex}$ et la preuve dans le sens restant procède par une induction directe sur h. □

Ce lemme permet de prouver qu'il est possible d'étendre un arbre candidat en preuve de non-bisimilarité en prouvant que toutes ses feuilles contiennent des états non-bisimilaires.

Corollaire 4.8. *Un cas particulier du lemme précédent est celui où tous les états des feuilles d'un arbre candidat produisent des sorties différentes.*

$$p \not\sim q \iff \forall (p',q') \in \mathsf{leaves}(h), \mathsf{out}(p') \neq \mathsf{out}(q').$$

Ces résultats n'ont rien de surprenant : ils procèdent de la définition même des arbres candidats, qui sont précisément des preuves de non-bisimilarité sans conditions sur les feuilles.

4.2.2 Conditions nécessaires et suffisantes à la réactivité d'un état composé

Les états résultants d'une composition séquentielle sont issus du produit cartésien des états de chaque composante. Etant donnés deux états composés, nous allons montrer comment toute

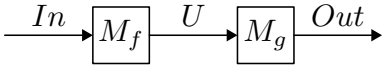
preuve de non-bisimilarité pour ces deux états est construite à partir de preuves de non-bisimilarité pour les états d'origine. Nous commençons par définir des ensembles d'arbres candidats particuliers, appelés respectivement *pré-compositions* et *post-compositions*. Nous montrons que ces arbres candidats sont effectivement des preuves de non-bisimilarité. Enfin, nous prouvons la réciproque de la proposition précédente, c'est-à-dire le fait que toute preuve de non-bisimilarité pour deux états composés s'exprime comme une *pré-composition* ou *une post-composition*. Nous supposons l'existence de types de données $In, U, Out \in \mathfrak{D}$, ainsi que de deux machines :

$$\begin{array}{rcl} M_f & = & \langle In, U, Q_f, E_f, \mathsf{out}_f, q_{i,f}\rangle, \\ M_g & = & \langle U, Out, Q_g, E_g, \mathsf{out}_g, q_{i,g}\rangle. \end{array}$$

Leur composition séquentielle $M_{g\circ f} = M_g \circ M_f$ a pour composantes :

$$M_{g\circ f} = \langle In, Out, Q_{g\circ f}, E_{g\circ f}, \mathsf{out}_{g\circ f}, (q_{i,f}, q_{i,g})\rangle.$$

Soient $p_f, q_f \in Q_f$ et $p_g, q_g \in Q_g$ respectivement deux états de M_f et deux états de M_g. Ces quatre états correspondent à deux états composés $(p_f, p_g) \in Q_{g\circ f}$ et $(q_f, q_g) \in Q_{g\circ f}$ dans la machine composée $M_{g\circ f}$. Nous étudions dans cette section les conditions sous lesquelles une preuve de non-bisimilarité $h_f \in \mathsf{c{-}ex}(p_f, q_f)$ donne lieu à une preuve de non-bisimilarité pour les états composés (p_f, p_g) et (q_f, q_g), et de même pour une preuve $h_g \in \mathsf{c{-}ex}(p_g, q_g)$. Rappelons que la composition séquentielle est illustrée par la figure qui suit :



Ceci justifie la terminologie de *pré* et *post-composition* utilisée dans la définition quelque peu technique qui suit. En termes informels, une pré-composition d'une preuve de non-bisimilarité $h \in \mathsf{c{-}ex}(p_g, q_g)$ correspond à l'action de décorer h avec des chemins d'exécution de M_f compatibles avec la structure de h. Tous les chemins de l'arbre que constitue une pré-composition doivent donc être des séquences d'exécution valides dans $M_g \circ M_f$. Les post-composition sont la contrepartie des pré-compositions, sur des preuves de $\mathsf{c{-}ex}(p_f, q_f)$.

De plus, nous rappellons que la notation $A + B$ correspond à l'union *disjointe* des ensembles A et B. Ceci nous sert à interpréter de façon constructive la disjonction logique $A \vee B$.

Définition 44 (Pré-composition et post-composition de preuves de non-bisimilarité)**.** *Les ensembles d'arbres candidats appelées pré et post-compositions sont définis par récursion mutuelle.*

- *Soit $h_g \in \mathsf{c{-}ex}(p_g, q_g)$ une preuve de non-bisimilarité pour p_g et q_g. L'ensemble des pré-compositions de h_g par deux états p_f et q_f est noté $\mathsf{PreC}(h_g, p_f, q_f)$. Par construction, nous avons $\mathsf{PreC}(h_g, p_f, q_f) \subseteq \mathsf{Candidates}_{M_{g\circ f}}((p_f, p_g),\ (q_f, q_g))$. $\mathsf{PreC}(h_g, p_f, q_f)$ est défini par induction sur h_g :*

$$\begin{array}{rcl} \mathsf{PreC}(\textsc{Base}(p_g, q_g), p_f, q_f) & = & \{\textsc{Base}((p_f, p_g),\ (q_f, q_g))\} \\ \mathsf{PreC}(\textsc{Ind}(p_g \xrightarrow{in_g} p'_g, s), p_f, q_f) & = & \varnothing \textit{ si } \mathsf{out}(p_f) \neq in_g \\ \mathsf{PreC}(\textsc{Ind}(p_g \xrightarrow{in_g} p'_g, s), p_f, q_f) & = & \\ \multicolumn{3}{c}{\left\{ \textsc{Ind}\left((p_f, p_g) \xrightarrow{in_f} (p'_f, p'_g), pre\right) \;\middle|\; p_f \xrightarrow{in_f} p'_f \in E_f \wedge pre \in \mathsf{PreC}(s) \right\}} \end{array}$$

$$\textit{où } \mathsf{PreC}(s) = \prod_{(q_f,q_g)\xrightarrow{in_f}(q'_f,q'_g)} \begin{array}{c} \mathsf{PreC}(s(q_g \xrightarrow{in_g} q'_g), p'_f, q'_f) \\ + \\ \{\mathsf{PostC}(h'_f, p'_g, q'_g) \mid h'_f \in \mathsf{c{-}ex}(p'_f, q'_f)\} \end{array}$$

- *Soit $h_f \in \mathsf{c{-}ex}(p_f, q_f)$ une preuve de non-bisimilarité pour p_f et q_f. L'ensemble des post-compositions de h_f par deux états p_g et q_g est noté $\mathsf{PostC}(h_f, p_g, q_g)$. Comme pour la pré-composition, nous avons $\mathsf{PostC}(h_f, p_g, q_g) \subseteq \mathsf{Candidates}_{M_{g \circ f}}((p_f, p_g),\ (q_f, q_g))$. $\mathsf{PostC}(h_f, p_g, q_g)$ est défini comme suit :*

$$\mathsf{PostC}(\textsc{Base}(p_f, q_f), p_g, q_g) = \left\{ \textsc{Ind}\left((p_f, p_g) \xrightarrow{in_f} (p'_f, p'_g), post\right) \;\middle|\; p_g \xrightarrow{\mathsf{out}(p_f)} p'_g \in E_g \wedge post \in \mathsf{PostC}_B \right\} \cup$$

$$\text{où } \mathsf{PostC}_B = \prod_{(q_f,q_g) \xrightarrow{in_f} (q'_f,q'_g)} \begin{matrix} \{\mathsf{PostC}(h'_f, p'_g, q'_g) \mid h'_f \in \mathsf{c{-}ex}(p'_f, q'_f)\} \\ + \\ \{\mathsf{PreC}(h'_g, p'_f, q'_f) \mid h'_g \in \mathsf{c{-}ex}(p'_g, q'_g)\} \end{matrix}$$

$$\mathsf{PostC}(\textsc{Ind}(p_f \xrightarrow{in_f} p'_f, s), p_g, q_g) = \left\{ \textsc{Ind}\left((p_f, p_g) \xrightarrow{in_f} (p'_f, p'_g), post\right) \;\middle|\; p_g \xrightarrow{\mathsf{out}(p_f)} p'_g \in E_g \wedge post \in \mathsf{PostC}(s) \right\}$$

$$\text{où } \mathsf{PostC}(s) = \prod_{(q_f,q_g) \xrightarrow{in_f} (q'_f,q'_g)} \begin{matrix} \mathsf{PostC}(s(q_f \xrightarrow{in_f} q'_f), p'_g, q'_g) \\ + \\ \{\mathsf{PreC}(h'_g, p'_f, q'_f) \mid h'_g \in \mathsf{c{-}ex}(p'_g, q'_g)\} \end{matrix}$$

Intuitivement, cette définition décrit les pré-compositions et post-compositions comme des "mille-feuilles" constitués de morceaux de contres-exemples provenant de $\mathsf{c{-}ex}(p_f, q_f)$ et de $\mathsf{c{-}ex}(p_g, q_g)$. Le théorème suivant affirme en substance que toute preuve de non-bisimilarité pour des états d'une composition séquentielle est décomposable en éléments de preuves de non-bisimilarité des états d'origine, soit comme une pré-composition, soit comme une post-composition.

Théorème 4.9 (Propagation des effets observables par composition séquentielle)**.**
Soient $(p_f, p_g) \in Q_{g \circ f}$ et $(q_f, q_g) \in Q_{g \circ f}$ deux états composés. L'équivalence suivante est vérifiée :

$$\mathsf{c{-}ex}((p_f, p_g), (q_f, q_g)) \neq \varnothing \iff \begin{matrix} (\exists h_f \in \mathsf{c{-}ex}(p_f, q_f), \mathsf{PostC}(h_f, p_g, q_g) \neq \varnothing) \\ \vee \\ (\exists h_g \in \mathsf{c{-}ex}(p_g, q_g), \mathsf{PreC}(h_g, p_f, q_f) \neq \varnothing) \end{matrix}$$

Nous commençons par un aperçu informel de la preuve de ce théorème.

– La preuve dans le sens (droite ⇒ gauche) procède par le simple fait que les pré-compositions et les post-compositions sont des arbres candidats dont les feuilles (cas BASE) produisent des sorties différentes. Cette propriété est suffisante pour en faire des contres-exemples à la bisimilarité, comme montré par un lemme précédent.
– Dans l'autre sens, nous procédons à une induction sur un contre-exemple exemple quelconque de $\mathsf{c{-}ex}((p_f, p_g), (q_f, q_g))$, et nous en épuisons tous les cas.

Démonstration.

- Nous commençons par prouver :

$$\exists h_f \in \mathsf{c{-}ex}(p_f, q_f), \mathsf{PostC}(h_f, p_g, q_g) \implies \mathsf{c{-}ex}((p_f, p_g), (q_f, q_g)) \neq \varnothing.$$

  Nous montrons que tout élément $h \in \mathsf{PostC}(h_f, p_g, q_g)$ est effectivement un élément de $\mathsf{c{-}ex}((p_f, p_g), (q_f, q_g))$. Observons d'abord que par définition de PostC, nous avons :

$$h \in \mathsf{Candidates}((p_f, p_g),\ (q_f, q_g)).$$

  L'ensemble des feuilles de h est calculé par $\mathsf{leaves}(h)$. Par définition de PreC et PostC, les feuilles sont toutes de la forme $\mathsf{leaves}(\mathsf{PreC}(\textsc{Base}(p_g, q_g), p_f, q_f)) = \{((p_f, p_g), (q_f, q_g))\}$. Puisque $\mathsf{out}(p_g) \neq \mathsf{out}(q_g)$, nous avons par définition de la composition séquentielle :

$$\mathsf{out}(p_f, p_g) \neq \mathsf{out}(q_f, q_g).$$

  Par application du Lemme 4.7, nous pouvons conclure $h \in \mathsf{c{-}ex}((p_f, p_g), (q_f, q_g))$. La preuve que les pré-compositions sont également des preuves de non-bisimilarité pour les états composés est similaire et est omise. Ceci clôt la première partie de la preuve.
- La preuve de :

$$\exists h_f \in \mathsf{c{-}ex}(p_f, q_f), \mathsf{PostC}(h_f, p_g, q_g) \implies \mathsf{c{-}ex}((p_f, p_g), (q_f, q_g)) \neq \varnothing$$

  est similaire au cas précédent.
- Il nous reste à prouver l'implication suivante :

$$\mathsf{c{-}ex}((p_f, p_g), (q_f, q_g)) \implies \begin{array}{l} (\exists h_f \in \mathsf{c{-}ex}(p_f, q_f), \mathsf{PostC}(h_f, p_g, q_g) \neq \varnothing) \\ \qquad\qquad \vee \\ (\exists h_g \in \mathsf{c{-}ex}(p_g, q_g), \mathsf{PreC}(h_g, p_f, q_f) \neq \varnothing)\,. \end{array}$$

  Nous procédons par induction sur l'hypothèse $h \in \mathsf{c{-}ex}((p_f, p_g), (q_f, q_g))$.
  - (Cas de base.) Nous avons $h = \textsc{Base}((p_f, p_g),\ (q_f, q_g))$. Par définition de la composition séquentielle, nous avons $\mathsf{out}(p_g) \neq \mathsf{out}(q_g)$. Il existe donc une preuve de la forme $\textsc{Base}(p_g, q_g) \in \mathsf{c{-}ex}(p_g, q_g)$, ce qui nous permet de conclure :

$$\mathsf{PreC}(\textsc{Base}(p_g, q_g), p_f, q_f) \neq \varnothing.$$

  - (Cas inductif.) Nous traitons le cas suivant :

$$h = \textsc{Ind}((p_f, p_g) \xrightarrow{in_f} (p'_f, p'_g), s).$$

    Comme illustré en Fig. 4.4, ce cas induit l'existence d'un ensemble de preuves :

$$\{h_{i,j} \in \mathsf{c{-}ex}((p'_f, p'_g), (q^i_f, q^j_g))\}.$$

    Nous procédons par analyse par cas sur les éléments de cet ensemble.
    - Si tous les éléments sont de la forme $h^{ij} = \mathsf{PostC}(h^j_f, p'_g, q^j_g)$ avec $h^j_f \in \mathsf{c{-}ex}(p'_f, q^i_f)$, nous pouvons construire directement une preuve $h_f \in \mathsf{c{-}ex}(p_f, q_f)$ telle que :

$$h_f = \textsc{Ind}(p_f \xrightarrow{in_f} p'_f, t); \quad \text{avec :}$$

$$t \in \left\{ q_f \xrightarrow{in_f} q^i_f \mapsto s((q_f, q_g) \xrightarrow{in_f} (q^i_f, q^j_g)) \mid (q_f, q_g) \xrightarrow{in_f} (q^i_f, q^j_g) \in E_{g \bullet f} \right\}.$$

- De même, si tous les éléments sont de la forme $h^{ij} = \mathsf{PreC}(h_g^j, p'_f, q_f^i)$ et tels que $h_g^j \in \mathsf{c{-}ex}(p'_g, q_g^j)$, nous pouvons construire une preuve $h_g \in \mathsf{c{-}ex}(p_g, q_g)$ telle que :

$$h_g = \textsc{Ind}(p_g \xrightarrow{in_f} p'_g, t);$$

où t est construite de façon symétrique au cas précédent.

- Enfin, il nous reste à traiter le cas où les pré-compositions et post-compositions coexistent. Nous procédons par analyse par cas (se référer à la figure 4.4 pour une meilleure compréhension).
  - Si, pour toute transition $q_f \xrightarrow{in_f} q_f^i \in E_f$, il existe une transition $q_g \xrightarrow{\mathsf{out}(q_f)} q_g^j \in E_g$ pour lesquelles $h^{ij} \in \mathsf{c{-}ex}((p'_f, p'_g), (q_f^i, q_g^j))$ soit telle que :

$$h^{ij} = \mathsf{PostC}(h_f^j, p'_g, q_g^j);$$

alors nous pouvons directement déduire de l'existence de h_f^j la proposition :

$$\forall q_f^i, q_f \xrightarrow{in_f} q_f^i \in E_f \implies p'_f \nsim q_f^i,$$

ce dont nous déduisons l'existence d'une preuve $h_f \in \mathsf{c{-}ex}(p_f, q_f)$ telle que :

$$h_f = \textsc{Ind}(p_f \xrightarrow{in_f} p'_f, t) \wedge \mathsf{PostC}(h_f, p_f, q_f) \neq \varnothing.$$

  - Dans l'autre cas, il existe une transition $q_f \xrightarrow{in_f} q_f^i \in E_f$ telle que pour toute transition $q_g \xrightarrow{\mathsf{out}(q_f)} q_g^j \in E_g$, nous ayons par élimination $h^{ij} = \mathsf{PreC}(h_g^i, p'_f, q_f^i)$. Si $\mathsf{out}(p_f) = \mathsf{out}(q_f)$, nous en déduisons l'existence d'une preuve $h_g \in \mathsf{c{-}ex}(p_g, q_g)$, de façon similaire au cas précédent. Sinon, nous en déduisons que nous somme dans le cas $h_f = \textsc{Base}(p_f, q_f)$ et le but suit par application du cas de base pour $\mathsf{PostC}(\textsc{Base}(p_f, q_f), p_g, q_g)$.

Le cas de la fermeture par symétrie (règle SYM) est omis car direct.

□

Nous pouvons maintenant tirer quelques conclusions de cette étude détaillée de la non-bisimilarité dans la composition séquentielle.

- Hors existence de pré-compositions et post-compositions, la réactivité n'est pas préservée par la composition séquentielle. Ces résultats s'étendent au délai de séparabilité, qui n'est pas conservé dans le cas général.
- En termes de méthode de vérification, cela signifie que pour vérifier que la composition de deux machines réponde bien à une spécification de réactivité en temps borné, il faut en toute généralité effectuer une recherche exhaustive sur l'espace d'état et les séparateurs. Dans la section qui suit, une méthode plus restreinte mais compositionnelle permettant de simplifier ce processus de vérification est proposée.

4.3 Effets observables sous la boucle de rétroaction

Bien que nous n'ayons pas procédé à une analyse détaillée du comportement des effets observables sous l'opération de boucle de rétroaction, nous conjecturons que la démarche adoptée dans le cas de la composition séquentielle est transposable au cas présent. Nous avons en

effet vu dans l'exemple 4.5 p. 98 qu'il est possible d'encoder la composition séquentielle comme une composition parallèle suivie d'une boucle de rétroaction : la composition séquentielle et la boucle de rétroaction sont clairement liées. Une piste que nous n'avons pas exploré est de montrer que toute boucle de rétroaction influant sur le délai de séparabilité peut être transformée en une composition séquentielle.

4.4 Sous-approximation de la réactivité

Cette section propose une solution à certains des problèmes rencontrés précédemment, à savoir :

1. le fait que les séparateurs ne garantissent pas d'effet observable,
2. la non-compositionnalité de la réactivité et des effets observables.

Nous procédons en restreignant notre attention aux cas où la réactivité, qui est une propriété en temps branchant des états, peut être réduite à une propriété en temps linéaire. Nous montrons ensuite la compositionnalité de cette approche, ainsi que son bon comportement sous l'opération de raffinement.

4.4.1 Sur-approximation et linéarisation des effets observables

Supposons l'existence d'une machine $M = \langle In, Out, Q, E, \mathsf{out}, q_i \rangle$. Soit $q \in Q$ un état de M. Nous allons définir une fonction associant à chaque temps logique (relativement à q) une sur-approximation des effets observables potentiellement générés par q ou n'importe quel état atteignable depuis q. Afin d'obtenir une méthode compositionnelle, nous devons nous assurer que cette sur-approximation sera stable par composition. Noter que q ne doit pas être nécessairement réactif.

Nous avons vu en Def. 8 p. 37 que les effets observables sont générés aux feuilles des preuves de non-bisimilarité. Un effet observable est donc nécessairement une différence entre deux mots de sortie générés par le même mot d'entrée depuis deux états distincts. Nous pouvons donc sur-approximer les effets observables par les différences entre les mots de sortie. Nous commençons par définir l'ensemble des états atteignables en t transitions.

Définition 45 (Etats t-atteignables)**.** *Soit $q \in Q$ un état et t un entier. L'ensemble des états atteignables en t transitions est défini par induction sur t. Nous désignerons ces états comme t-atteignables depuis q, ce qui sera formellement noté* $\mathsf{reachable}(q, t)$.

$$\begin{array}{lcl} \mathsf{reachable}(q, 0) & = & \{q\} \\ \mathsf{reachable}(q, t+1) & = & \bigcup_{q \xrightarrow{in} q' \in E} \mathsf{reachable}(q', t) \end{array}$$

Une sur-approximation brutale des effets observables dont l'occurrence survient t unités de temps après l'état q est de considérer tous les couples de symboles de sortie possibles pour les états t-atteignables. En considérant $t = 0, 1, 2, \ldots$, nous obtenons une séqence de sur-approximation des effets observables depuis un état donné. Si l'espace d'état est fini, cette sur-approximation est représentable de façon finie.

Définition 46 (Linéarisation des effets observables d'un état)**.** *Soit $q \in Q$ un état de M et $t \in \mathbb{N}$ un entier représentant un temps logique relatif à q. Soient $S = \{\mathsf{out}(q_t) \mid q_t \in \mathsf{reachable}(q,t)\}$ les symboles de sortie potentiellement générés après t transitions. La sur-approximation des effets observables au temps logique t pour p et q est notée $\mathsf{OEseq}(p,q,t)$ et est définie ainsi :*

$$\mathsf{OEseq}(q,t) = S \times S - Id,$$

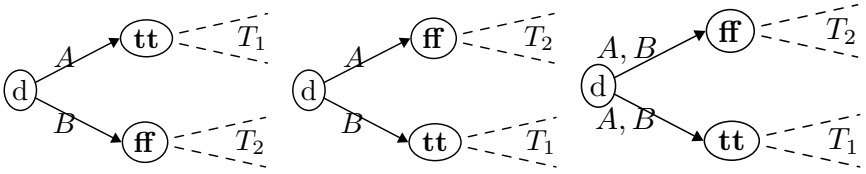
où Id est la relation identité (les éléments de la forme (a,a) ne pouvant être des effets observables).

4.4.2 Sous-approximation et linéarisation des paires séparantes

Afin d'obtenir une méthode compositionnelle, il nous faut résoudre deux problèmes :
- déterminer un moyen de réduire la propriété de réactivité d'un état au fait que la sur-approximation des effets observables est inclue dans une sous-approximation adéquate des paires séparantes ;
- faire en sorte que l'occurrence d'un effet observable associé à une paire séparante ne dépende pas du contexte et soit donc garanti pour toute entrée.

La sous-approximation évoquée ci-dessus sera calculée comme l'intersection d'un certain type de paire séparante pour tous les états atteignables en un temps donné. Considérons maintenant deux états composés $(p_f, p_g) \in Q_{g \circ f}$ et $(q_f, q_g) \in Q_{g \circ f}$ tels que $h_f \in \mathsf{c{-}ex}(p_f, q_f)$ avec $h_f = \textsc{Base}(p_f, q_f)$. Nous voulons définir une condition suffisante assurant que l'effet observable induit par h_f est bien pris en compte par M_g, c'est-à-dire par p_g et q_g. Nous commençons par noter que prendre l'intersection des paires séparantes de q_g et p_g n'est pas suffisant, comme illustré par l'exemple qui suit.

Exemple 4.10 : Etudions les états en Fig. 4.5, dans lesquels seules les données en sorties sont décrites et où les noms des états sont omis. Les états 1 et 2 sont symétriques et ont tous les deux (A, B) comme paire séparante. Toutefois, (A, B) n'est pas une paire séparante pour l'*union* des deux états.



(a) Etat 1 (b) Etat 2 (c) Union des états 1 et 2

FIGURE 4.5 – Union d'états

Transposons cette situation à notre cas d'étude. Soient :

$$P = \{p'_g \mid p_g \xrightarrow{\mathsf{out}(p_f)} p'_g\}/\sim \quad \text{et} \quad Q = \{q'_g \mid q_g \xrightarrow{\mathsf{out}(q_f)} q'_g\}/\sim$$

les ensembles d'états (quotientés par bisimilarité) atteignables depuis p_g par $\mathsf{out}(p_f)$ (resp. atteignables depuis q_g par $\mathsf{out}(q_f)$). Le fait que $(\mathsf{out}(p_f), \mathsf{out}(q_f)) \in \mathsf{SepPairs}(p_g) \bigcap \mathsf{SepPairs}(q_g)$ n'est pas contradictoire avec $P = Q$, i.e. avec l'absence d'effet observable. Nous développons donc une notion de paire séparante valide pour un ensemble d'états, et qui garantisse la compositionnalité. Nous les désignons "paires fortement séparantes".

Définition 47 (Paire fortement séparante pour un ensemble d'états)**.** *Soit $S \subseteq Q$ un ensemble d'états. L'ensemble des paires fortement séparantes de S est noté* $\mathsf{StrongSepPairs}(S)$*, et est défini ainsi :*

$$(a_1, a_2) \in \mathsf{StrongSepPairs}(S) \iff \\ \forall q_1, q_2 \in S, \forall q_1', q_2' \in Q, q_1 \xrightarrow{a_1} q_1' \in E \ \wedge \ q_2 \xrightarrow{a_2} q_2' \in E \implies q_1' \mathrel{)\!(} q_2'.$$

Nous pouvons maintenant définir une linéarisation des paires fortement séparantes d'un état q à un temps t.

Définition 48 (Linéarisation des paires séparantes stables d'un état)**.** *Soit $q \in Q$ un état et t un entier. Notre sous-approximation des paires séparantes au temps t sera notée* $\mathsf{SSPseq}(q, t)$*, et calculée comme suit :*

$$\mathsf{SSPseq}(q, t) = \mathsf{StrongSepPairs}(\mathsf{reachable}(q, t)).$$

4.4.3 Critère de réactivité

Dans cette section, nous supposons l'existence de deux machines :

$$\begin{aligned} M_f &= \langle In, U, Q_f, E_f, \mathsf{out}_f, q_{i,f} \rangle, \\ M_g &= \langle U, Out, Q_g, E_g, \mathsf{out}_g, q_{i,g} \rangle. \end{aligned}$$

ainsi que de leur composition séquentielle

$$M_{g \circ f} = M_g \circ M_f = \langle In, Out, Q_{g \circ f}, E_{g \circ f}, \mathsf{out}_{g \circ f}, (q_{i,f}, q_{i,g}) \rangle.$$

A l'aide des fonctions OEseq et SSPseq, nous pouvons établir un critère permettant de sous-approximer la réactivité d'un état composé (q_f, q_g) : si les effets observables de q_f au temps t sont inclus dans les paires séparantes stables de q_g au temps t, alors la composition est réactive. Ce critère est exprimé par le théorème suivant.

Théorème 4.11 (Réactivité approchée)**.** *Soient $q_f \in Q_f$ et $q_g \in Q_g$ deux états et $(q_f, q_g) \in Q_{g \circ f}$ leur composition séquentielle. La propriété suivante est vérifiée :*

$$(\forall t \in \mathbb{N}, \mathsf{OEseq}(q_f, t) \subseteq \mathsf{SSPseq}(q_g, t) \wedge \mathsf{reactive}(q_f)) \implies \mathsf{reactive}(q_f, q_g).$$

Démonstration. Dans l'intérêt de la concision, nous omettrons la preuve des cas symétriques dans ce qui suit. Nous devons montrer l'existence d'une paire séparante (a_1, a_2) pour (q_f, q_g). La propriété correspondante est :

$$\exists (q_f^1, q_g^1) \in Q_{g \circ f}, (q_f, q_g) \xrightarrow{a_1} (q_f^1, q_g^1) \in E_{g \circ f} \wedge \forall (q_f^2, q_g^2) \in Q_{g \circ f}, \\ (q_f, q_g) \xrightarrow{a_2} (q_f^2, q_g^2) \in E_{g \circ f} \implies (q_f^1, q_g^1) \nsim (q_f^2, q_g^2).$$

Nous avons comme hypothèse de départ :

$$\forall t \in \mathbb{N}, \mathsf{OEseq}(q_f, t) \subseteq \mathsf{SSPseq}(q_g, t) \wedge \mathsf{reactive}(q_f).$$

Choisissons une paire séparante quelconque $(a_1, a_2) \in \mathsf{SepPairs}(q_f)$. Par définition, nous avons :

$$\exists q_f^1 \in Q_f, q_f \xrightarrow{a_1} q_f^1 \in E_f \wedge \forall q_f^2 \in Q_f, q_f \xrightarrow{a_2} q_f^2 \in E_f \implies q_f^1 \nsim q_f^2.$$

Nous pouvons donc affirmer l'existence d'un contre-exemple $h_f \in \mathsf{c{-}ex}(q_f^1, q_f^2)$ pour tous les (q_f^1, q_f^2). Il nous reste à montrer l'existence d'une post-composition de h_f. Nous commençons par construire un arbre *candidat* $post_f \in \mathsf{Candidates}_{M_{g \circ f}}((q_f^1, q_g^1), (q_f^2, q_g^2))$, puis nous montrons comment l'étendre en une post-composition de h_f en utilisant l'hypothèse de départ pour remplacer les feuilles de $post_f$ par des arbres preuves de non-bisimilarité.

1) Construction de $post_f$. Nous calculons $post_f$ par récursion sur h_f, à l'aide d'une fonction decorate prenant en argument une preuve de non-bisimilarité sur M_f et deux états de Q_g :

$$post_f = \mathsf{decorate}(h_f, q_g^1, q_g^2).$$

Cette fonction étiquette *arbitrairement* h_f par des états de Q_g atteignables depuis q_g^1 et q_g^2.

$$\begin{array}{lcl}
\mathsf{decorate}(\textsc{Base}(p_f, q_f),\ p_g,\ q_g) & = & \textsc{Base}((p_f, p_g),\ (q_f, q_g)) \\
\mathsf{decorate}(\textsc{Sym}(h_g),\ p_g,\ q_g) & = & \textsc{Sym}((decorate(h_g),\ q_g,\ p_g)\textsc{Base}((p_f, p_g),\ (q_f, q_g)) \\
\mathsf{decorate}(\textsc{Ind}(p_f \xrightarrow{in_f} p'_f, s),\ p_g,\ q_g) & = & \textsc{Ind}((p_f, p_g) \xrightarrow{in_f} (p'_f, p'_g), s')
\end{array}$$

avec $p_g \xrightarrow{\mathsf{out}(p_f)} p'_g \in E_g$ quelconque, et

$s' = (q_f, q_g) \xrightarrow{in_f} (q'_f, q'_g) \mapsto \mathsf{decorate}(s'(q_f \xrightarrow{in_f} q'_f), p'_g, q'_g)$

2) Extension de $post_f$ et conclusion. Nous construisons un élément de $\mathsf{PostC}(h_f, q_g^1, q_g^2)$ par induction sur $post_f$ (le cas symétrique est omis).

- (Cas de base.) Nous avons $post_f = \textsc{Base}((q_f^1, q_g^1),\ (q_f^2, q_g^2))$ avec $h_f = \textsc{Base}(q_f^1, q_f^2)$. Par définition de OEseq, nous déduisons :

  $$\exists t, (\mathsf{out}(q_f^1), \mathsf{out}(q_f^2)) \in \mathsf{OEseq}(q_f, t).$$

  L'application de l'hypothèse de départ nous donne :

  $$(\mathsf{out}(q_f^1), \mathsf{out}(q_f^2)) \in \mathsf{SSPseq}(q_g, t),$$

  i.e. $(\mathsf{out}(q_f^1), \mathsf{out}(q_f^2)) \in \mathsf{StrongSepPairs}(\{q_g^1;\ q_g^2\})$. Par définition, nous obtenons (les cas symétriques sont traités de façon identique) :

  $$\exists {q'}_g^1 \in Q_g, q_g^1 \xrightarrow{\mathsf{out}(q_f^1)} {q'}_g^1 \in E_g \ \wedge\ \forall {q'}_g^2 \in Q_g, q_g^2 \xrightarrow{\mathsf{out}(q_f^2)} {q'}_g^2 \in E_g \implies {q'}_g^1 \between {q'}_g^2.$$

  Puisque ${q'}_g^1 \between {q'}_g^2$, nous avons directement $\exists h'_g \in \mathsf{c{-}ex}({q'}_g^1, {q'}_g^2)$ et plus particulièrement :

  $\forall p_1, p_2 \in Q_f, \mathsf{PreC}(h'_g, p_1, p_2) \neq \varnothing$. *(Ici apparaît l'indépendance vis-à-vis du contexte.)*

  Nous pouvons donc affirmer (par définition de PostC, dans le cas inductif) qu'il existe :

  $$\textsc{Ind}((q_f^1, q_g^1) \xrightarrow{in_f} ({q'}_f^1, {q'}_g^1), s) \in \mathsf{PostC}(h_f, q_g^1, q_g^2),$$

  où $q_f^1 \xrightarrow{in_f} q_f^2 \in E_f$ est une transition quelconque.
- (Cas inductif.) Nous avons :

  $$post_f = \textsc{Ind}((q_f^1, q_g^1) \xrightarrow{in_f} ({q'}_f^1, {q'}_g^1), s).$$

  Par application de l'hypothèse d'induction sur les états atteignables par $post_f$, ces derniers sont non bisimilaires. Nous pouvons directement en déduire que $post_f \in \mathsf{PostC}(h_f, q_g^1, q_g^2)$, ce qui conclut la preuve.

□

4.5 Compositionnalité de la sous-approximation

Dans la section précédente, nous avons défini un ensemble de conditions *suffisantes* pour s'assurer de la réactivité d'un état composé et permettant d'éviter l'exploration de l'espace d'état de la machine composée. Cet ensemble d'informations est donné dans notre cas par les fonctions OEseq et SSPseq, calculant respectivement la séquence des effets observables et la séquence des paires fortement séparantes.

Déterminer ces fonctions à partir des machines est aussi coûteux en termes calculatoires que de décider directement de la réactivité des états composés. Une approche qui recalculerait à chaque composition les fonctions OEseq et SSPseq à partir de l'espace d'état composé ne présenterait aucun avantage. Afin d'offrir une approche compositionnelle, il nous faut impérativement pouvoir recalculer OEseq et SSPseq pour les états composés à partir des fonctions correspondantes de chaque composante.

Nous nous plaçons dans les hypothèses de la Section 4.4.3 p. 109, en admettant l'existence de machines M_f, M_g et de leur composition séquentielle $M_{g\circ f}$. Le théorème suivant exprime que notre formalisme exhibe la propriété de compositionnalité recherchée.

Théorème 4.12 (Compositionalité de la linéarisation)**.** *Soient $q_f \in Q_f$ et $q_g \in Q_g$ deux états de respectivement M_f et M_g, et $(q_f, q_g) \in Q_{g\circ f}$ un état composé tels que les conditions nécessaires à l'application du Théorème 4.11 soient vérifiées, c'est-à-dire. :*

$$\forall t \in \mathbb{N}, \mathsf{OEseq}(q_f, t) \subseteq \mathsf{SSPseq}(q_g, t) \wedge \mathsf{reactive}(q_f).$$

Les deux propriétés suivantes sont alors vérifiées :

1. $\forall t, (o_1, o_2) \in \mathsf{OEseq}((q_f, q_g),\ t) \implies (o_1, o_2) \in \mathsf{OEseq}(q_g, t)$,
2. $\forall t, (a_1, a_2) \in \mathsf{SSPseq}(q_f,\ t) \implies (a_1, a_2) \in \mathsf{SSPseq}((q_f, q_g), t)$.

En d'autres termes, la sur-approximation des effets observables de (q_f, q_g) est incluse dans celle de q_g. Réciproquement, les paires fortement séparantes de q_f sont comprises dans celles de (q_f, q_g).

Démonstration. Soit $t \in \mathbb{N}$ quelconque.

1. Posons $(o_1, o_2) \in \mathsf{OEseq}((q_f, q_g),\ t)$. Par définition de OEseq, il existe deux états (q_f^1, q_g^1) et (q_f^2, q_g^2) atteignables depuis (q_f, q_g) en t transitions. Nous pouvons obtenir par simple projection de la seconde composante deux séquences d'exécutions correspondantes dans M_g, ce qui implique $(o_1, o_2) \in \mathsf{OEseq}(q_g,\ t)$.
2. Supposons $(a_1, a_2) \in \mathsf{SSPseq}(q_f,\ t)$. Nous avons par définition de SSPseq :

$$\begin{aligned}\forall q_f^1, q_f^2 \in \mathsf{reachable}(q_f, t), \exists {q'}_f^1 \in Q_f,\\ q_f^1 \xrightarrow{a_1} {q'}_f^1 \in E_f \wedge \forall {q'}_f^2 \in Q_f, q_f^2 \xrightarrow{a_2} {q'}_f^2 \in E_f \implies {q'}_f^1 \between {q'}_f^2 \vee\\ q_f^1 \xrightarrow{a_2} {q'}_f^1 \in E_f \wedge \forall {q'}_f^2 \in Q_f, q_f^2 \xrightarrow{a_1} {q'}_f^2 \in E_f \implies {q'}_f^1 \between {q'}_f^2.\end{aligned}$$

   Soient $(q_f^1, q_g^1) \in Q_{g\circ f}$ et $(q_f^2, q_g^2) \in Q_{g\circ f}$ deux états composés atteignables en t transitions depuis (q_f, q_g). Il nous faut prouver :

$$\begin{aligned}\exists({q'}_f^1, {q'}_g^1) \in Q_{g\circ f}, (q_f^1, q_g^1) \xrightarrow{a_1} ({q'}_f^1, {q'}_g^1) \in E_{g\circ f} \wedge \forall({q'}_f^2, {q'}_g^2) \in Q_{g\circ f},\\ (q_f^2, q_g^2) \xrightarrow{a_2} ({q'}_f^2, {q'}_g^2) \implies ({q'}_f^1, {q'}_g^1) \between ({q'}_f^2, {q'}_g^2) \vee\\ \exists({q'}_f^1, {q'}_g^1) \in Q_{g\circ f}, (q_f^1, q_g^1) \xrightarrow{a_2} ({q'}_f^1, {q'}_g^1) \in E_{g\circ f} \wedge \forall({q'}_f^2, {q'}_g^2) \in Q_{g\circ f},\\ (q_f^2, q_g^2) \xrightarrow{a_1} ({q'}_f^2, {q'}_g^2) \implies ({q'}_f^1, {q'}_g^1) \between ({q'}_f^2, {q'}_g^2)\end{aligned}$$

Il suffit de montrer $q'^1_f \;)\!(\; q'^2_f \implies (q'^1_f, q'^1_g) \;)\!(\; (q'^2_f, q'^2_g)$. Soit $w \in In^\omega$ un mot infini quelconque. Par hypothèse, $w = s.w'$ où $s \in \mathsf{S}_*(q'^1_f, q'^2_f)$. Considérons deux séquences d'exécution quelconques induites par s :

$$(q'^1_f, q'^1_g) = (q'^{1,1}_f, q'^{1,1}_g) \xrightarrow{s[0]} \ldots (q'^{1,|s|}_f, q'^{1,|s|}_g) \text{ et}$$
$$(q'^2_f, q'^2_g) = (q'^{2,1}_f, q'^{2,1}_g) \xrightarrow{s[0]} \ldots (q'^{2,|s|}_f, q'^{2,|s|}_g).$$

Par définition des pseudo-séparateurs, il existe un indice i tel que $(\mathsf{out}(q'^{1,i}_f), \mathsf{out}(q'^{2,i}_f))$ est un effet observable. Par hypothèse de départ, cet effet observable est inclus dans les paires fortement séparantes de $(q'^{1,i}_g, q'^{2,i}_g)$. Nous en déduisons :

$$\forall q'^{1,|s|+1}_g, q'^{2,|s|+1}_g \in Q_g,$$
$$q'^{1,|s|}_g \xrightarrow{\mathsf{out}(q'^{1,|s|}_f)} q'^{1,|s|+1}_g \in E_g \;\wedge\; q'^{2,|s|}_g \xrightarrow{\mathsf{out}(q'^{2,|s|}_f)} q'^{2,|s|+1}_g \in E_g \implies q'^{1,|s|+1}_g \;)\!(\; q'^{2,|s|+1}_g$$

ce qui nous permet de conclure que $(q'^1_f, q'^1_g) \;)\!(\; (q'^2_f, q'^2_g)$.

□

4.6 Conservation de la sous-approximation par simulation inverse

Nous avons vu au Chapitre 3 que l'opération de raffinement est une instance de la relation de simulation inverse, comme étudiée au Chapitre 2, Sec. 2.4 p. 42. Afin d'intégrer le développement de systèmes temps-réels par compositions successives dans une démarche de développement par raffinement, deux conditions doivent être vérifiées :

1. les propriétés d'intérêt (réactivité ou temps de séparabilité) doivent être conservées par raffinement,
2. le raffinement doit également préserver les structures et informations permettant la compositionnalité.

Le premier point a été l'objet de l'étude de la section 3.4. Le second point est l'objet de cette section. La compositionnalité de la sous-approximation de la réactivité présentée dans les sections précédentes repose sur les deux fonctions OEseq et SSPseq, associées à chaque état. Nous montrons que ces deux fonctions sont préservées par raffinement et n'ont pas besoin d'être recalculées à partir de l'espace d'état de la machine. Ce fait est exprimé par le lemme suivant.

Lemme 4.13 (Préservation de OEseq et SSPseq par simulation inverse)**.** *Soit une machine :*

$$M = \langle In, Out, Q, E, \mathsf{out}, q_0 \rangle$$

et $q \in Q$ un état quelconque. La propriété suivante est vérifiée :

$$\forall q' \precsim q, \forall t \in \mathbb{N}, \mathsf{OEseq}(q', t) \subseteq \mathsf{OEseq}(q, t) \wedge \mathsf{SSPseq}(q, t) \subseteq \mathsf{SSPseq}(q', t).$$

Démonstration. Soit q' tel que $q' \precsim q$ et $t \in \mathbb{N}$ un instant logique quelconque. Nous prouvons $\mathsf{OEseq}(q', t) \subseteq \mathsf{OEseq}(q, t)$ puis $\mathsf{SSPseq}(q, t) \subseteq \mathsf{SSPseq}(q', t)$.

1) Preuve de $\mathsf{OEseq}(q',t) \subseteq \mathsf{OEseq}(q,t)$. Soit $(o_1,o_2) \in \mathsf{OEseq}(q',t)$. Par définition, il existe deux états q_1 et q_2 atteignables en t transitions tels que $o_1 = \mathsf{out}(q_1)$ et $o_2 = \mathsf{out}(q_2)$. Par définition de la simulation, il existe des séquences d'exécution correspondantes depuis q et nous avons donc $(o_1,o_2) \in \mathsf{OEseq}(q,t)$.

2) Preuve de $\mathsf{SSPseq}(q,t) \subseteq \mathsf{SSPseq}(q',t)$. Soit $(a_1,a_2) \in \mathsf{SSPseq}(q,t)$. Nous avons par définition de $\mathsf{SSPseq}(q,t)$:

$$\forall q_1,q_2 \in \mathsf{reachable}(q,t), \forall q_1',q_2' \in Q, q_1 \xrightarrow{a_1} q_1' \in E \ \wedge \ q_2 \xrightarrow{a_2} q_2' \in E \implies q_1' \ )\!(\ q_2'.$$

Il est aisé de prouver que $\mathsf{reachable}(q',t)/\sim \ \subseteq \ \mathsf{reachable}(q,t)/\sim$ (la simulation restreint les séquences d'exécutions possibles). Qui plus est, les pseudo-séparateurs sont précisément la plus grande classe de mots garantissant un effet observable pour toute entrée et *conservés par raffinement*. Par conséquent, nous obtenons directement $(a_1,a_2) \in \mathsf{SSPseq}(q',t)$. □

Nous pouvons directement déduire de ce lemme la préservation de la compositionnalité par raffinement.

Proposition 4.14. *Soient les machines M_f, M_g et $M_{g\circ f}$ comme définies en Sec. 4.4.3. Soient $q_f \in Q_f$ et $q_g \in Q_g$ deux états et $(q_f,q_g) \in Q_{g\circ f}$ leur composition séquentielle. Supposons que les hypothèses nécessaires à prouver la réactivité de (q_f,q_g) soient vérifiées :*

$$\forall t \in \mathbb{N}, \mathsf{OEseq}(q_f,t) \subseteq \mathsf{SSPseq}(q_g,t) \wedge \mathsf{reactive}(q_f).$$

Ces hypothèses sont préservées par raffinement :

$$\forall q_f' \precsim q_f, \forall q_g' \precsim q_g, \forall t \in \mathbb{N}, \mathsf{OEseq}(q_f',t) \subseteq \mathsf{SSPseq}(q_g',t) \wedge \mathsf{reactive}(q_f').$$

Démonstration. La preuve procède par simple application du Lemme 4.13. □

Notons pour conclure que la méthode approchée que nous proposons est extrèmement brutale et présente en pratique un faible intérêt, puisque elle impose des restrictions drastiques sur le comportement des états : tous les états que l'on veut garantir réactifs doivent en quelque sorte être réceptifs à tous les effets observables possibles. Il faut noter que l'intérêt de cette section réside non pas dans l'application directe de son contenu, mais dans la perspective de proposer de méthodes approchées arbitrairement plus fines. La leçon à tirer de ces développements est que pour garantir le délai de séparabilité de façon compositionnelle, il faut avoir à l'avance une bonne approximation de ce que la machine doit calculer, afin de donner des contraintes les plus précises possibles sur les paires séparantes et les effets observables. Finalement, il s'agit d'un compromis absolument classique, et presque tautologique : plus la spécification est précise et localisée, moins le système risque d'être surcontraint. Ceci est particulèrement vrai avec le délai de séparabilité, une spécification par elle-même très contraignante sur les aspects fonctionnels et temporels.

4.7 Synthèse

Afin de permettre l'étude du développement par compositions successives, nous avons défini trois primitives : la composition séquentielle, la boucle de rétroaction et la composition parallèle. Ce chapitre a permis l'étude des propriétés de la composition séquentielle et plus particulièrement le comportement de la propriété de réactivité par composition séquentielle. Nous

avons montré par quelques contre-exemples la non-conservation de la réactivité par cette opération. Nous avons ensuite donné les conditions nécessaires et suffisantes à la réactivité d'un état composé, en montrant comment toute preuve de non-bisimilarité dans une machine composée s'écrit en termes de preuves de non-bisimilarité de ses composantes.

Nous avons ensuite proposé une solution au problème de la non-compositionnalité de la réactivité. Notre méthode repose sur une linéarisation des effets observables et des paires séparantes, afin de réduire la vérification de la réactivité à une forme d'inclusion de langages. Cette méthode a été montrée apte à la conception incrémentale, dans la mesure où les linéarisations pour les états composés peuvent être déduites directement des linéarisations de chaque composante. Enfin, nous avons montré la conservation de ces mêmes linéarisations par raffinement.

A partir d'un ensemble de contraintes induites par la compositionnalité et le raffinement, nous avons défini une sous-approximation de la réactivité. Le point d'achoppement est le calcul des fonctions OEseq et SSPseq sur la machine initiale. En pratique, ce calcul ne peut être effectué que lorsque le flot d'information est déjà exprimé dans la machine. L'exemple de sous-approximation que nous avons donné dans ce chapitre n'est pas facilement utilisable durant les premières phases de développement. Il représente un point extrême dans le spectre des sous-approximations, et peut servir de point de départ pour des variations plus subtiles et plus à même d'être mises en oeuvre.

Une approche raisonnable serait par exemple de laisser au concepteur le soin de préciser quels sont les états devant être réactifs, et pour ces derniers, de préciser en plus du délai de séparabilité, un ensemble de paires séparantes devant être préservées. Enfin, plutôt que de chercher la conservation du délai de séparabilité pour toute simulation inverse, la relation de raffinement pourrait être restreinte de façon à ne pas permettre la supression des paires séparantes spécifiées. Ceci rejoint les considérations sur la conservation du délai de séparabilité dans le cas pessimiste, évoquées à la fin du chapitre 3 en Sec. 3.4.3.2 p. 87.

5 Conclusion

L'objectif de cette thèse était de concevoir un formalisme de développement par composition et par raffinement permettant de prouver des contraintes temporelles sur des systèmes conçus avec OASIS. Nous procédons à un examen critique de nos résultats et proposons quelques perspectives de recherche.

5.1 Résultats

Le point de départ fut de décider d'un modèle formel apte à représenter les systèmes développés avec OASIS. Une étude observationnelle de ces systèmes nous a permis de les classer dans la famille des systèmes synchrones faibles. Une sémantique opérationnelle a également été définie pour PsyC. Les systèmes OASIS étant représentables par des machines de Moore, le problème originel a été ainsi circonscrit à l'étude de ces machines. Dans le chapitre 2, nous avons par la suite identifié le délai de séparabilité d'un état comme le temps de trajet d'une information en entrée à travers la machine depuis cet état. Afin de donner à nos résultats un caractère général, nous avons également considéré l'information contenue dans la structure branchante de nos machines de Moore non-déterministes, et pas seulement l'information contenue dans les traces d'entrée/sortie. Enfin, une étude de la conservation du délai de séparabilité a montré que cette information n'est en général pas conservée par la relation de simulation. Des conditions nécessaires et suffisantes à la conservation du délai de séparabilité sont déduites de ces résultats. La conclusion de cette étude préliminaire montre que seuls les mots induisant des langages de sortie disjoints (appelés ici pseudo-séparateurs) permettent de garantir un effet observable pour une entrée initiale. Or, cette propriété sur les langages de sortie ignore totalement la structure branchante des machines. L'information contenue dans la structure branchante des machines n'est donc pas conservée par simulation, ce qui rétrospectivement est naturel.

Dans le chapitre 3, nous avons étudié les propriétés d'une variante des machines de Moore non-déterministes où les données ont une structure de treillis, dans le dessein de nous en servir comme formalisme de spécification haut-niveau. Nous avons donné les conditions sous lesquelles le délai de séparabilité sur ces machines permet d'approximer correctement le délai de séparabilité d'une machine simple. Par la suite, nous avons prouvé un résultat nous permettant d'étendre l'étude du délai de séparabilité au cas de machines ayant plusieurs entrées et plusieurs sorties et ce de façon efficace. Ces résultats sont collectés dans un formalisme de développement incrémental par raffinement où cette dernière relation est définie comme étant la simulation. La preuve de préservation du délai de séparabilité par raffinement est apportée en considérant uniquement les pseudo-séparateurs dans le calcul de ce délai.

Le chapitre 4 achève notre étude du délai de séparabilité par l'investigation des propriétés

conservées par composition. Nous avons défini les opérations de composition séquentielle (ou composition parallèle communicante), de composition parallèle non-communicante et de boucle de rétroaction sur les machines de Moore non-déterministes. Une étude détaillée de la conservation du délai de séparabilité sous l'opération de composition séquentielle a été entreprise, montrant sa non-conservation. Ces résultats sont appliqués à la définition de contraintes sur les machines permettant de garantir la préservation du délai de séparabilité par composition séquentielle *et* raffinement. Ces contraintes reposent sur la réduction du problème d'origine à un problème de vérification d'inclusion de langage.

5.2 Perspectives

Nous entrevoyons de nombreux axes de recherche potentiels. Nous en proposons quelques-uns, certains directement liés au contenu de ce manuscrit et d'autres plus généraux.

A très court terme, il nous semble primordial de traiter un exemple d'application de taille raisonnable. Nos expériences préliminaires montrent que cette tâche est difficilement réalisable sans support logiciel [35, 34]. De plus, au cours de notre étude, nous ne nous sommes pas intéressés au fait de donner des bornes de complexité précises pour des problèmes cependant décidables, comme le fait de savoir si deux états sont séparables en un délai t. Nous conjecturons que ces problèmes sont de complexité exponentielle en la taille de l'espace d'état et en t. Nous avons répondu aux problèmes de non-conservation du délai de séparabilité par abstraction ou par raffinement en cherchant des sous-approximations de la notion de séparabilité. Il serait intéressant de procéder à l'étude duale, c'est à dire chercher les notions d'abstraction et de raffinement qui préservent le délai de séparabilité général. Enfin, il est en théorie possible de générer automatiquement du code PsyC (pour OASIS) à partir d'une machine de Moore déterministe. Cependant, les machines de Moore sont beaucoup trop basiques pour permettre la génération de code efficace. PsyC permet l'échange de données entre tâches par flots de données et par envois de messages ; et les transitions temporelles peuvent être de plus d'une seule unité de temps. Générer du code PsyC directement à partir d'une machine de Moore entraînerait probablement un trop grand nombre d'appels à la couche système d'OASIS. Deux solutions sont à priori envisageables : effectuer certaines analyses afin d'optimiser la génération de code, ou enrichir le formalisme afin de pouvoir directement exprimer les communications à haut-niveau.

De façon plus générale, nous nous sommes cantonnés au cas de systèmes synchrones particuliers. Il serait intéressant d'étendre dans un premier temps nos définitions au cas des machines de Mealy (ce devrait être direct), de façon à pouvoir traiter le modèle synchrone fort. Enfin, l'idée de considérer les effets observables d'une entrée sur les sorties semble applicable à tout modèle des systèmes réactifs, y-compris les modèles asynchrones, comme par exemple les automates temporisés. A cet effet, il nous apparaît important de généraliser nos outils. Une perspective de recherche particulièrement pertinente et intéressante serait d'établir des liens formels entre nos développements et la théorie de l'information de Shannon. Une machine de Moore déterministe peut effectivement être considérée comme un canal de communication bruité. Dans le cas non-déterministe, nous postulons qu'une partie de nos développements peut être utilisée pour définir l'*entropie de Shannon* entre une entrée et une sortie (au sens large), ce qui permettrait de quantifier *numériquement* le nombre de bits de l'entrée effectivement utilisés pour produire la sortie. Ceci ouvre la possibilité de vérifier qu'un système de contrôle commande utilise *au moins* une certaine quantité d'information.

Bibliographie

[1] P. A. Abdulla and J. Deneux. Designing safe, reliable systems using scade, 2004.

[2] S. Abramsky. Retracing some paths in process algebra. In *CONCUR '96 : Concurrency Theory, 7th International Conference*, pages 1–17. Springer-Verlag, 1996.

[3] S. Abramsky, S. Gay, and R. Nagarajan. Interaction categories and the foundations of typed concurrent programming. In M. Broy, editor, *Proceedings of the 1994 Marktoberdorf Summer School on Deductive Program Design*, pages 35–113. Springer-Verlag, 1996.

[4] J.-R. Abrial. Steam-boiler control specification problem. In *Formal Methods for Industrial Applications, Specifying and Programming the Steam Boiler Control (the book grow out of a Dagstuhl Seminar, June 1995).*, pages 500–509, London, UK, 1996. Springer-Verlag.

[5] P. Aczel. *Non-Well-Founded Sets*. Csli Lecture Notes, 1988.

[6] L. d. Alfaro, T. A. Henzinger, and M. Stoelinga. Timed interfaces. In *Proceedings of the Second International Conference on Embedded Software*, EMSOFT '02, pages 108–122, London, UK, 2002. Springer-Verlag.

[7] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126 :183–235, 1994.

[8] P. Amagbégnon, L. Besnard, and P. Le Guernic. Implementation of the data-flow synchronous language signal. In *Proceedings of the ACM SIGPLAN 1995 conference on Programming language design and implementation*, PLDI '95, pages 163–173, New York, NY, USA, 1995. ACM.

[9] S. Anantharaman and G. Hains. A synchronous bisimulation based approach for information flow analysis. In *IN PROC. OF THE THIRD WORKSHOP AVOCS03, SOUTHAMPTON (UK)*, 2003.

[10] D. Austry and G. Boudol. Algebre de processus et synchronisation. Research Report RR-0187, INRIA, 1983.

[11] A. Basu, M. Bozga, and J. Sifakis. Modeling heterogeneous real-time components in bip. In *SEFM '06 : Proceedings of the Fourth IEEE International Conference on Software Engineering and Formal Methods*, pages 3–12. IEEE Computer Society, 2006.

[12] S. Bensalem, M. Bozga, T.-H. Nguyen, and J. Sifakis. D-finder : A tool for compositional deadlock detection and verification. In *Proceedings of the 21st International Conference on Computer Aided Verification*, CAV '09, pages 614–619, Berlin, Heidelberg, 2009. Springer-Verlag.

[13] G. Berry. The constructive semantics of pure esterel, 1999.

[14] G. Berry and L. Cosserat. The esterel synchronous programming language and its mathematical semantics. In *Seminar on Concurrency, Carnegie-Mellon University*, pages 389–448, London, UK, 1985. Springer-Verlag.

[15] J. Bhasker and R. Chandra. *Static Timing Analysis for Nanometer Designs*. Springer, 2009.

[16] A. Bohannon, B. C. Pierce, V. Sjöberg, S. Weirich, and S. Zdancewic. Reactive noninterference. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 79–90, New York, NY, USA, 2009. ACM.

[17] F. Boniol. Processus réactifs communicants synchrones.

[18] M. M. Bonsangue, J. Rutten, and R. Silva. A. : Coalgebraic logic and synthesis of mealy machines. Technical report.

[19] T. Bourke. Modelling and programming embedded controllers with timed automata and synchronous languages, 2009.

[20] F. Boussinot. Reactive c : An extension of c to program reactive systems, 1991.

[21] F. Boussinot and R. de Simone. The sl synchronous language. *IEEE Trans. on Software Engineering*, 22 :256–266, 1996.

[22] M. Bozga, J.-C. Fernandez, L. Ghirvu, S. Graf, J. pierre Krimm, and L. Mounier. If : An intermediate representation and validation environment for timed asynchronous systems, 1999.

[23] M. Bozga, S. Graf, I. Ober, I. Ober, and J. Sifakis. The if toolset. In M. Bernardo and F. Corradini, editors, *Formal Methods for the Design of Real-Time Systems*, volume 3185 of *Lecture Notes in Computer Science*, pages 131–132. Springer Berlin / Heidelberg, 2004.

[24] J. R. Buechi. On a Decision Method in Restricted Second-Order Arithmetic. In *International Congress on Logic, Methodology, and Philosophy of Science*, pages 1–11. Stanford University Press, 1962.

[25] P. Bulychev, T. Chatain, A. David, and K. G. Larsen. Efficient on-the-fly algorithm for checking alternating timed simulation.

[26] P. Caspi, N. Scaife, C. Sofronis, and S. Tripakis. Semantics-preserving multitask implementation of synchronous programs. *ACM Trans. Embed. Comput. Syst.*, 7 :15 :1–15 :40, January 2008.
[27] D. Clark, S. Hunt, and P. Malacaria. A static analysis for quantifying information flow in a simple imperative language. *J. Comput. Secur.*, 15 :321–371, August 2007.
[28] E. M. Clarke, D. E. Long, and K. L. Mcmillan. Compositional model checking, 1999.
[29] P. Cousot and R. Cousot. Abstract interpretation : a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, 1977. ACM Press, New York, NY.
[30] P. Cousot and R. Cousot. Inductive definitions, semantics and abstract interpretations. In *POPL '92 : Proceedings of the 19th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 83–94, New York, NY, USA, 1992. ACM.
[31] A. David, K. G. Larsen, A. Legay, U. Nyman, and A. Wasowski. Timed i/o automata : a complete specification theory for real-time systems. In *Proceedings of the 13th ACM international conference on Hybrid systems : computation and control*, HSCC '10, pages 91–100, New York, NY, USA, 2010. ACM.
[32] V. David, J. Delcoigne, E. Leret, A. Ourghanlian, P. Hilsenkopf, and P. Paris. Safety properties ensured by the oasis model for safety critical real-time systems. In *SAFECOMP*, 1998.
[33] M. Erne, J. Koslowski, A. Melton, and G. Strecker. A primer on galois connections, 1992.
[34] I. Garnier, C. Aussaguès, V. David, and G. Vidal-Naquet. Formally ensuring time constraints in a development process. In *Proceedings of the VVPS workshop*, 2011.
[35] I. Garnier, C. Aussaguès, V. David, and G. Vidal-Naquet. On the reaction time of some synchronous systems. In *Proceedings of the ICE workshop*, 2011.
[36] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types (Cambridge Tracts in Theoretical Computer Science)*. Cambridge University Press, 1989.
[37] J. Goguen and J. Meseguer. Security policies and security models. In I. C. S. Press, editor, *Proc of IEEE Symposium on Security and Privacy*, pages 11–20, April 1982.
[38] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous dataflow programming language lustre. *Proceedings of the IEEE*, 79(9) :1305–1320, September 1991.
[39] H. H. Hansen, D. Costa, and J. RUtten. Synthesis of mealy machines using derivatives. In *In : Proceedings of CMCS 2006. ENTCS*, page 2006. Elsevier, 2006.
[40] W. H. Hesselink. Simulation refinement for concurrency verification. *Sci. Comput. Program.*, 76 :739–755, September 2011.
[41] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8) :666–677, 1978.
[42] B. Jacobs and J. Rutten. A tutorial on (co)algebras and (co)induction. *EATCS Bulletin*, 62 :222–259, 1997.
[43] A. Joyal, R. Street, and D. Verity. Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society*, 119(03) :447–468, 1996.
[44] G. Kahn. Natural semantics. Rapport de recherche RR-0601, INRIA, 1987.
[45] D. K. Kaynar, N. Lynch, R. Segala, and F. Vaandrager. Timed i/o automata : A mathematical framework for modeling and analyzing real-time systems. *Real-Time Systems Symposium, IEEE International*, 0 :166, 2003.
[46] T. Le Berre. *Spécification formelle de systèmes temps réel répartis par une approche flots de données à contraintes temporelles*. Thèse de doctorat, Institut National Polytechnique de Toulouse, Toulouse, France, mars 2010.
[47] D. Lee and M. Yannakakis. Principles and methods of testing finite state machines-a survey. *Proceedings of the IEEE*, 84(8) :1090–1123, 1996.
[48] X. Leroy and H. Grall. Coinductive big-step operational semantics. *CoRR*, abs/0808.0586, 2008.
[49] C. Loiseaux and S. Graf. Property preserving abstractions for the verification of concurrent systems. In *Formal Methods in System Design*, 1995.
[50] L. Mandel and M. Pouzet. ReactiveML, a reactive extension to ML. In *Proceedings of 7th ACM SIGPLAN International conference on Principles and Practice of Declarative Programming (PPDP'05)*, 2005.
[51] F. Maraninchi. The argos language : Graphical representation of automata and description of reactive systems. In *IEEE Workshop on Visual Languages*, Kobe, Japan, oct 1991.

[52] F. Maraninchi and Y. Rémond. Argos : an automaton-based synchronous language. *Computer Languages*, 2001.
[53] Mathworks. . `http://www.mathworks.se/products/simulink/`, 2011.
[54] R. Milner. *A Calculus of Communicating Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982.
[55] R. Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25(3) :267 – 310, 1983.
[56] E. F. Moore. Gedanken Experiments on Sequential Machines. In *Automata Studies*, pages 129–153. Princeton U., 1956.
[57] X. Nicollin and J. Sifakis. An overview and synthesis on timed process algebras, 1991.
[58] P. Pettersson. Modelling and verification of real-time systems using timed automata : Theory and practice, 1999.
[59] G. D. Plotkin. A structural approach to operational semantics. *J. Log. Algebr. Program.*, 60-61 :17–139, 2004.
[60] T. H. R. Alur. Logics and Models of Real Time : A Survey. *Lecture Notes in Computer Science*, 1992.
[61] C. Ramchandani. Analysis of asynchronous concurrent systems by timed petri nets. Technical report, Cambridge, MA, USA, 1974.
[62] J. C. Reynolds. Idealized algol and its specification logic. *ALGOL-like Languages, Volume 1*, pages 125–156, 1997.
[63] F. C. O.-H. Roux. From time petri nets to timed automata, 2004.
[64] K. Schneider. A verified hardware synthesis for esterel programs. In *in Proceedings of the International IFIP Workshop on Distributed and Parallel Embedded Systems (DIPES*, pages 205–214. Kluwer, B.V, 2000.
[65] G. Smith. Principles of secure information flow analysis. In *Malware Detection*, pages 297–307. Springer-Verlag, 2007.
[66] Y. Sorel. From modeling/simulation with scilab/scicos to optimized distributed embedded real-time implementation with syndex. In *Proceedings of the International Workshop On Scilab and Open Source Software Engineering, SOSSE'05*, Wuhan, China, 2005.
[67] M. Stannett. Simulation testing of automata. *Formal Aspects of Computing*, 18 :31–41, 2006. 10.1007/s00165-005-0080-y.
[68] R. J. van Glabbeek. The linear time-branching time spectrum (extended abstract). In *CONCUR*, pages 278–297, 1990.
[69] N. Wirth. Program development by stepwise refinement. *Commun. ACM*, 14 :221–227, April 1971.
[70] S. Yovine. Kronos : A verification tool for real-time systems. (kronos user's manual release 2.2). *International Journal on Software Tools for Technology Transfer*, 1 :123–133, 1997.

A

Sémantique opérationnelle

Le point de départ de notre étude a été le langage PsyC, utilisé au sein d'OASIS pour programmer les processus synchrones nommés *agents*. Le langage PsyC ajoute au langage C principalement trois éléments : une notion de temps formelle, une approche multi-tâche et la possibilité de communiquer entre ces tâches. Parmi ses traits caractéristiques, nous pouvons noter les suivants.

- Le programme est structuré en un nombre statiquement défini d'agents s'exécutant de façon concurrente.
- Chaque agent est programmé dans un sous-ensemble de C dépourvu de récursion. L'allocation "dynamique" de mémoire ne peut se faire que par l'utilisation d'un tas statiquement alloué et de taille fixe. La récursion est interdite.
- L'exécution de chaque agent est temporellement découpée en intervalles spécifiés par l'utilisateur par l'instruction **advance(t)**, où t est la taille de l'intervalle temporel entre l'instruction **advance(t)** considérée et la *précédente* instruction **advance**. En d'autre termes, un bloc de code :
  advance(x) ;
  codeA ;
  advance(10)
  permet à codeA de s'exécuter dans une fenêtre temporelle de 10 pas d'horloge. Le respect de ces contraintes temporelles est vérifié dynamiquement par le système d'exploitation.
- Les agents communiquent par variables temporelles (flot de données) ou par messages dont le contenu est daté.
- Les exécutions ne terminent pas.

Ces éléments nous ont servi de point de départ pour définir une sémantique formelle pour PsyALGOL, une simplification de PsyC en conservant les principaux traits inspirée d'une restriction d'Idealized Algol [62] au premier ordre (puisqu'il n'y a pas de fonctions ni de procédures).

A.1 Syntaxe et typage des agents

A.1.1 Notations

Une variable x est libre dans une expression si elle n'est "pas déclarée". Si e est une expression, $FV(e)$ est l'ensemble des variables libres dans cette expression. On utilisera la notation $[R/y]E$ pour dénoter l'expression E ou toutes les occurrences **libres** de y sont remplacées par R.

A.1.2 Types

Les types sont les suivants :

$$\begin{array}{rcl} \tau & ::= & \textbf{int} \mid \textbf{bool} \\ \sigma & ::= & \textbf{comm} \mid \textbf{var}(\tau) \mid \textbf{exp}(\tau) \end{array}$$

On identifiera les types atomiques avec les ensembles qu'ils représentent. On définit également des valeurs par défaut pour les types atomiques :

$$\begin{array}{rcl} default_{\textbf{int}} & = & 0 \\ default_{\textbf{bool}} & = & \textbf{ff} \end{array}$$

Les types τ correspondent à des valeurs, le type **var(**τ**)** à des références ("pointeurs") sur des valeurs de type τ et le type **exp(**τ**)** correspond à une expression retournant une valeur de type τ. Le type **comm** est le type des commandes ne retournant pas de valeur. Ceci est une approximation : en lieu et place de **comm**, on peut enrichir la classe τ avec un type singleton **unit** (correspondant à une valeur unique qui serait **skip**) et définir **comm** = **exp(unit)**. Cette équivalence sera utilisée pour la règle de typage de **new**, dans la suite.

A.1.3 Syntaxe et règles de typage pour les agents

Un système est composé d'un ensemble d'agents communicants. Un agent est un programme réactif de type synchrone. Les intevalles temporels séparant chaque activation sont spécifiés par le programmeur. A chaque activation, il lit les données en entrées (produites par les autres agents) et rend disponible les données calculées précédemment. Les données sont de deux genres : des données consommables à volonté (nommées "variables temporelles", VT en abrégé) et des messages consommables une fois exactement. En PsyC, le fait de ne pas consommer un message pendant sa fenêtre de validité est considéré comme une erreur détectée durant l'exécution.

A.1.3.1 Interface d'un agent

L'interface d'un agent est formée de son interface en entrée et de son interface en sortie.

Interface en entrée. L'interface en entrée est constituée des signatures pour un ensemble fini de VT et un ensemble fini de boites aux lettres (BAL) pour les messages.

Soit $\mathcal{I}(vt)$ le nombre de variables temporelles en entrée et $\mathcal{I}(msg)$ le nombre de BAL. A chaque VT en entrée est associée un type τ_i^{invt} où $i \in [1; \mathcal{I}(vt)]$ est l'indice associé à la variable temporelle. De même, à chaque BAL est associée un type τ_i^{inmsg} (où $i \in [1; \mathcal{I}(msg)]$ est l'indice associé à la boite).

Interface en sortie. L'interface en sortie est constituée des signatures pour un ensemble fini de VT, et un ensemble fini de ports de sortie pour les messages.

Soit $\mathcal{O}(vt)$ le nombre de variables temporelles en sortie, et $\mathcal{O}(msg)$ le nombre de ports de sortie. A chaque VT en sortie est associée un type τ_i^{outvt} où $i \in [1; \mathcal{O}(vt)]$. De même, à chaque port est associé un type τ_i^{outmsg} avec $i \in [1; \mathcal{O}(msg)]$.

A.1.3.2 Règles de formation et de typage des agents

La définition des termes est donnée de façon inductive à l'aide de règles d'inférences portant sur des jugements de la forme $\Gamma \vdash M : \sigma$, signifiant "dans le contexte Γ, le programme M a le type σ". Un contexte Γ est de la forme $x_1 : \sigma_1, \ldots, x_n : \sigma_k$, et sert à associer à des variables x_i leur type σ_i. On suppose l'existence de Var, un ensemble dénombrable de variables.

$$\frac{}{\Gamma \vdash \textbf{skip} : \textbf{comm}} \qquad \frac{}{\Gamma, x : \sigma \vdash x : \sigma \qquad x \in Var}$$

$$\frac{b \in \{\textbf{tt}, \textbf{ff}\}}{\Gamma \vdash b : \textbf{exp(bool)}} \qquad \frac{n \in \mathbb{N}}{\Gamma \vdash n : \textbf{exp(int)}}$$

$$\frac{\Gamma \vdash V : \textbf{var}(\tau)}{\Gamma \vdash !V : \textbf{exp}(\tau)} \qquad \frac{\Gamma, x : \textbf{var}(\tau_0) \vdash C : \textbf{exp}(\tau_1)}{\Gamma \vdash \textbf{new}\ x\ : \tau_0\ \textbf{in}\ C : \textbf{exp}(\tau_1)}$$

$$\frac{\Gamma \vdash V : \textbf{var}(\tau) \qquad \Gamma \vdash E : \textbf{exp}(\tau)}{\Gamma \vdash V := E : \textbf{comm}} \qquad \frac{\Gamma \vdash A_0 : \textbf{comm} \qquad \Gamma \vdash A_1 : \sigma}{\Gamma \vdash A_0; A_1 : \sigma}$$

$$\frac{\Gamma \vdash A_0 : \textbf{exp(bool)} \qquad \Gamma \vdash A_0 : \sigma \qquad \Gamma \vdash A_1 : \sigma}{\Gamma \vdash \textbf{if}\ A_0\ \textbf{then}\ A_1\ \textbf{else}\ A_2 : \sigma} \qquad \frac{\Gamma \vdash A_0 : \textbf{exp(bool)} \qquad \Gamma \vdash A_0 : \textbf{comm}}{\Gamma \vdash \textbf{while}\ A_0\ \textbf{do}\ A_1\ \textbf{done} : \sigma}$$

$$\frac{\Gamma \vdash A_1 : \textbf{exp}(\tau_1^{outvt}) \qquad \ldots \qquad \Gamma \vdash A_{\mathcal{O}(vt)} : \textbf{exp}(\tau_{\mathcal{O}(vt)}^{outvt})}{\Gamma \vdash \textbf{advance}(n,\ A_1, \ldots, A_{\mathcal{O}(vt)}) : \textbf{comm} \qquad n > 0} \qquad \frac{}{\Gamma \vdash \textbf{get}_i : \textbf{exp}(\tau_i^{invt}) \qquad i \in [1; \mathcal{I}(vt)]}$$

$$\frac{\Gamma \vdash A : \textbf{exp}(\tau_i^{outmsg})}{\Gamma \vdash \textbf{send}_i(A, n) : \textbf{comm} \qquad i \in [1; \mathcal{O}(msg)], n \in \mathbb{N}^+} \qquad \frac{}{\Gamma \vdash \textbf{take}_i : \textbf{exp}(\tau_i^{inmsg}) \qquad i \in [1; \mathcal{I}(msg)]}$$

$$\frac{}{\Gamma \vdash \textbf{count}_i : \textbf{exp(int)} \qquad i \in [1; \mathcal{I}(msg)]}$$

Un terme A est *bien typé* dans Γ si il existe un arbre de dérivation utilisant les règles ci-dessus et dont la racine est $\Gamma \vdash A : \sigma$. Soit $Prog(\Gamma)$ l'ensemble des termes bien typés dans le contexte Γ et $Prog$ l'ensemble des termes tels qu'il existe un contexte permettant de bien les typer.

Notons que l'instruction **advance** de PsyALGOL est différente de celle de PsyC. En PsyC, **advance** est uniquement paramétrée par la taille de l'intervalle temporel, alors qu'en PsyALGOL elle est également paramétrée par le nouvel état observable des variables temporelles de l'agent. Nous décrirons plus en détails pourquoi notre présentation est équivalente à celle de PsyC lors de la définition de la sémantique opérationnelle.

A.2 Sémantique opérationnelle des agents

A.2.1 Préliminaire à la définition de la sémantique opérationnelle

Nous allons désormais procéder à la définition de la sémantique opérationnelle du langage. Il s'agit de rajouter à un langage impératif simple et idéalisé, les traits synchrones apportés par

le modèle **OASIS**. On omet les procédures, sans perte d'expressivité.

Certaines particularités doivent être soulignées pour comprendre la formalisation donnée ci-dessous. L'évaluation d'un fragment de programme réactif en PsyALGOL peut se dérouler de trois manières distinctes :

1. Le fragment peut s'évaluer et terminer son exécution, donnant une **valeur**.
2. Le fragment peut diverger (par exemple, une boucle infinie sans **advance**).
3. Le fragment peut s'évaluer sans jamais terminer mais rester "productif"(boucle infinie avec **advance**).

Ces trois possibilités sont capturées dans la sémantique opérationnelle à petits pas ("small-step structured operational semantics", Plotkin [59]), mais cette sémantique est beaucoup moins pratique à utiliser que la sémantique naturelle ("big-step", Kahn [44]). Dans le cas de la sémantique naturelle, les cas 1 d'une part et $2, 3$ d'autre parts, doivent être définis de façon différente (Cousot & Cousot [30], Leroy & Grall [48]). Certaines règles d'inférence devront être interprétées de manière *inductive* (cas 1), d'autres de manière coinductive (cas 2 et 3).

Qui plus est, le fait qu'une exécution normale puisse ne jamais se terminer rend nécessaire l'utilisation d'une notion d'observable autre que la simple valeur de retour. Nous utiliserons une notion de *trace* potentiellement infinie pour caractériser le résultat de l'exécution d'un agent.

Le point le plus complexe de la modélisation que nous donnons est la gestion des messages, dont nous donnons ici une description informelle. Cette description justifiera la définition de certaines opérations sur les boites aux lettres, qui seront utilisées lors de la définition de la sémantique, section A.2.3.

Lors de l'envoi d'un message avec la commande $\mathbf{send}_i(A, n)$, l'utilisateur spécifie le FIFO de sortie i, la valeur à envoyer (produit de l'évaluation de A) et un **temps d'apparition** n. Ce temps est donné relativement au point d'activation spécifié par l' **advance** précédant l'évaluation de **send**. Rien n'empêche l'utilisateur de définir le programme $\mathbf{send}_i(v, 3); \mathbf{send}_i(v, 1)$ - en d'autres termes, l'ordre d'évaluation ne correspond *pas* à l'ordre d'envoi des messages ...

Qui plus est, un message peut devenir visible **avant** le prochain **advance**, comme dans l'expression $\mathbf{send}_i(v_1, 3); \mathbf{advance}(5,\ v_2)$. On serait tenté de penser que la primitive **send** effectue un **advance** caché, mais ce n'est pas exact : les variables temporelles ne sont mises à jour qu'aux **advance** et pas aux **send**. Sans les messages, la définition de la trace d'un agent PsyC se définit comme un "homomorphisme" : la trace est calculée directement selon la structure de l'arbre d'évaluation. Ce n'est plus le cas avec les messages. Il faut inclure dans les agents un moyen de se souvenir quels sont les messages devenant visibles à telle ou telle date.

Cette complexité, inhérente à l'objet que nous tentons de modéliser, rend la définition d'une sémantique opérationnelle relativement difficile. Nous allons modéliser ces envois de messages ainsi : à chaque **send**, la valeur et la date de visibilité sera stockée dans une file d'attente (cf. configuration d'un agent). A chaque **advance**, les messages censés être visibles avant cet **advance** sont en quelque sorte "préfixés" à la trace de l'agent, comme si ils avaient été envoyés avant. Les autres messages voient leur date de visibilité décroître de la valeur de l'**advance**.

A.2.2 Représentation de l'état d'un programme PsyALGOL en cours d'exécution

L'exécution d'un programme PsyALGOL requiert plusieurs éléments, en plus de l'expression PsyALGOL à réduire. Nous définissons ici chacune de ces composantes :

– l'état de ses variables internes,
– la valeur des variables temporelles en entrée et en sortie,
– l'ensemble des messages reçus ainsi que ceux non encore rendus visibles.

A.2.2.1 Représentation des variables déclarées et substitutions

Les variables déclarées du programme sont représentées par une fonction associant leur valeur à leur nom. Cette fonction est appelée un *store*. Un V*-store* est une fonction de la forme :

$$st : V \rightarrow \textbf{int} \cup \textbf{bool},$$

où $V \subset Var$ est un ensemble des variables formant le domaine de st. Soit $Store(V)$ l'ensemble des V*-stores*. Si $st \in Store(V)$, on note $s \restriction x$ le *store* obtenu en restreignant le domaine de s et $(s \mid x \mapsto n)$ celui où s est mis à jour en associant à x la valeur n (ou en écrasant l'ancienne valeur de x). Si $st \in Store(V)$ et $x \in V$, $st(x)$ est la valeur associée à x. Par exemple, $S = \{x \mapsto 3, y \mapsto \textbf{tt}\ , z \mapsto 10\}$ est un $\{x, y, z\}$ – $store$. On a $S \restriction y = \{x \mapsto 3, z \mapsto 10\}$ et $(S \mid y \mapsto \textbf{ff}\ ) = \{x \mapsto 3, y \mapsto \textbf{ff}\ , z \mapsto 10\}$.

A.2.2.2 Représentation des files de messages

Notre définition de la sémantique opérationnelle des agents nécessite l'utilisation de files d'attentes (FIFOs). Une file d'attente sur un ensemble support Σ (d'où proviennent les éléments) est un mot fini sur l'alphabet Σ. On dénote $FIFO(\Sigma) = \Sigma^*$ l'ensemble des files d'attentes sur Σ. La file d'attente vide est $\epsilon \in \Sigma^*$. Les files de messages contiendront des couples (m, t) constitués du message en lui-même m et d'une durée $t \in \mathbb{Z}$. Soit un ensemble quelconque Σ. Nous définissons quelques fonctions sur les files de $FIFO(\Sigma \times \mathbb{Z})$.

Insertion d'un message dans une file. La fonction d'insertion d'un message dans une file se contente d'ajouter le symbole muni de sa date de visibilité en tête de file.

$$\begin{array}{l} \mathsf{insert} : \Sigma \times \mathbb{Z} \times FIFO(\Sigma \times \mathbb{Z}) \rightarrow FIFO(\Sigma \times \mathbb{Z}) \\ \mathsf{insert}(m, t, s) = s.(m, t) \end{array}$$

Extraction d'un message dans une file. L'extraction d'un message est symétrique à l'insertion.

$$\begin{array}{l} \mathsf{extract} : FIFO(\Sigma) \rightarrow \Sigma \times FIFO(\Sigma) \\ \mathsf{extract}(\epsilon)\ \textit{indéfinie} \\ \mathsf{extract}(m.s) = (m, s) \end{array}$$

Effacement des messages ne répondant pas à un prédicat. Nous pouvons définir une fonction d'ordre supérieur effaçant tous les éléments d'une file ne répondant pas à un prédicat donné en argument. Cette fonction est définie par récursion sur la file.

$$\begin{array}{lcl} \mathsf{filter} : (\Sigma \times \mathbb{Z} \rightarrow \mathbb{B}) \times FIFO(\Sigma \times \mathbb{Z}) \rightarrow FIFO(\Sigma \times \mathbb{Z}) & & \\ \mathsf{filter}(pred, \epsilon) & = & \epsilon \\ \mathsf{filter}(pred, (m, t).s) & = & (m, t).\mathsf{filter}(pred, s) \text{ si } pred(m, t) \\ \mathsf{filter}(pred, (m, t).s) & = & \mathsf{filter}(pred, s) \text{ sinon} \end{array}$$

A.2.2.3 Traces

Nous allons dans la suite de ce document définir la trace générée par un agent, c'est-à-dire la suite de ses observations. La définition de ces traces est donnée ici.

A chaque *tick*, on observe d'un agent l'état de ses variables temporelles et les messages devenus visibles. Chaque message n'est visible dans une trace que pendant une unité de temps (même si en pratique il pourra rester plus longtemps dans la boite aux lettres de l'agent récepteur).

Nous allons également définir une opération de fusion sur les traces finies, qui servira à donner la sémantique de l'envoi de messages (voir la définition de la sémantique opérationnelle pour **advance**).

Alphabet des traces de sortie. L'alphabet est un couple contenant d'une part les valeurs courantes des variables temporelles et d'autre part les messages visibles dans chaque port d'émission de message. L'alphabet pour les variables temporelles est noté Σ_{vt} et est défini par :

$$\Sigma_{vt} = \prod_{i \in [1;\mathcal{O}(vt)]} \tau_i^{outvt}.$$

Un symbole de Σ_{vt} est donc un vecteur $\overrightarrow{vt} = (v_1,\ v_2, \ldots,\ v_{\mathcal{O}(vt)})$ où $v_i \in \tau_i^{outvt}$. L'alphabet des messages est noté Σ_{msg}, défini par l'union disjointe de leurs ensembles d'origines pour chaque BAL de sortie :

$$\Sigma_{msg} = \coprod_{i \in [1;\mathcal{O}(msg)]} \tau_i^{outmsg}.$$

Σ_{msg} est constitué d'éléments $inj_i(m) \in \coprod_{i \in [1;\mathcal{O}(msg)]} \tau_i^{outmsg}$, où $m \in \tau_i^{outmsg}$. Puisque plusieurs messages peuvent êtres émis sur des ports de sortie arbitraires au même temps logique, nous devons considérer à chaque temps logique une séquence de tels messages (l'ordre d'envoi des messages est pris en compte dans PsyC). L'alphabet des traces est alors défini comme un couple constitué d'un symbole de Σ_{vt} et d'une séquence sur Σ_{msg} :

$$\Sigma_{tr} = \Sigma_{vt} \times \Sigma_{msg}^*.$$

Cette définition donne lieu a un alphabet infini à cause de la seconde composante mais nous verrons qu'en pratique, la longueur des files de messages en sortie est bornée. Une trace est un mot fini ou infini sur cet alphabet Σ_{tr}.

Messages contenus dans une trace. Nous aurons besoin d'accéder aux messages contenus dans une trace. En l'occurrence, il s'agit de collecter la seconde composante de chaque symbole et de concaténer le tout. La fonction correspondante est définie récursivement sur les traces finies :

$$\begin{array}{lcl} \mathsf{messages} & : & \Sigma_{tr}^* \to \Sigma_{msg}^* \\ \mathsf{messages}(\epsilon) & = & \epsilon \\ \mathsf{messages}((\overrightarrow{vt}, s).tr) & = & s.\mathsf{messages}(tr) \end{array}$$

A.2.2.4 Représentation du système extérieur à l'agent

En OASIS, un agent ne s'exécute pas isolément. Il échange des données avec les autres agents par le biais du système d'exploitation dédié. Ce passage par le système d'exploitation est

un artefact de l'implémentation. Comme nous l'avons montré dans ce manuscrit, un ensemble communicant d'"agents" peut être créé à l'aide d'un nombre minimal de primitives de composition. Dans la définition de la sémantique opérationnelle de PsyALGOL, nous modélisons l'environnement du programme en cours d'évaluation par un objet défini ci-dessous.

Soient les alphabets d'entrée Σ_{in} et de sortie Σ_{out} correspondant à l'interface du programme en cours d'exécution :

$$\Sigma_{in} = \prod_{i\in[1;\mathcal{I}(vt)]} \tau_i^{invt} \times \left(\coprod_{i\in[1;\mathcal{I}(msg)]} \tau_i^{inmsg} \right)^*,$$

$$\Sigma_{out} = \Sigma_{vt} \times \Sigma_{msg}^* = \Sigma_{tr}.$$

Un contexte interagissant avec cet agent est un objet appartenant à un ensemble $Context$ solution de l'équation [42] :

$$Context \simeq \Sigma_{in} \times (\Sigma_{out} \to Context).$$

Cette spécification décrit un programme où les interfaces d'entrée et de sortie du programme considéré ont été inversées. Noter que ce programme faisant office d'environnement peut lui-même être issus d'une composition d'autres sous-programmes. Soit $cx \in Context$ un contexte. La signature d'un contexte supporte deux opérations, correspondant aux fonctions de projection.

$$\begin{array}{lll} \mathsf{obs} : Context \to \Sigma_{in} & & \mathsf{next} : Context \times \Sigma_{out} \to Context \\ \mathsf{obs}(cx) = \pi_1(cx) & & \mathsf{next}(out, cx) = \pi_2(s)(out) \end{array}$$

La fonction obs renvoie l'état observable du contexte, correspondant par définition à l'entrée du programme en cours d'évaluation. La fonction next calcule le contexte au tick d'horloge suivant à partir de la réaction du programme en cours d'évaluation.

En pratique, un agent peut progresser de plusieurs ticks d'horloge d'un seul coup. Nous étendons la fonction next de façon à accepter des traces (finies et non-vides). La définition de next_{tr} est par induction sur les traces. Elle accepte une trace de Σ_{tr}^* et génère en sus de l'état du système une trace de Σ_{in}^* de même longueur (le troisième argument est un simple accumulateur, initialement égal à ϵ).

$$\begin{array}{lcl} \mathsf{next}_{tr} : \Sigma_{tr}^* \times Context \times \Sigma_{in}^* \to Context \times \Sigma_{in}^* & & \\ \mathsf{next}_{tr}(x.t, syst, acc) & = & \mathsf{next}_{tr}(t, \mathsf{next}(x, syst), acc.\mathsf{obs}(syst)) \\ \mathsf{next}_{tr}(x.\epsilon, s, acc) & = & (\mathsf{next}(x, s), acc.\mathsf{obs}(syst)) \end{array}$$

A.2.2.5 Configuration d'un programme

Nous allons définir une relation d'évaluation sur les programmes. Un programme à un point d'exécution donné sera muni de sa *configuration*, représentant l'état mémoire du système. Une configuration est un quintuplet :

$$\langle st, cx, inbox, \overrightarrow{outvts}, outbox \rangle,$$

dont les composantes sont décrites ci-dessous.

1. $st \in Store$ est un *store*, représentant l'état des variables du programme ;

2. $cx \in Context$ est le contexte dans lequel se déroule l'exécution ;
3. $inbox$ est la boite aux lettres en entrée, représentée par un FIFO contenant des couples (valeur, temps avant péremption) ;
4. $\overrightarrow{outvts}$ est l'état des variables temporelles du programme en sortie ;
5. $outbox$ est une file d'attente à priori non bornée stockant les messages envoyés et en attente d'être consommés.

Soit $Config$ l'ensemble des configurations. Les composantes 2, 3, 4 ne sont modifiées que lors de l'exécution d'un **advance**. Afin d'alléger les notations, nous utiliserons la notation (par exemple) $\langle st, X \rangle$ où X recouvre les composantes de 2 à 5.

A.2.3 Sémantique opérationnelle "big-step"

Dans la suite, les règles inductives n'ont qu'une seule barre et celles coinductives en ont deux (on suit la convention de [48]). Soit $\mathcal{V}$ = **int** $\cup$ **bool** $\cup$ {**skip**} l'ensemble des valeurs. Nous allons définir deux relations : $\Rightarrow \subseteq (Config \times Prog) \times (Config \times \mathcal{V})$ et $\underset{\infty}{\Downarrow} \subseteq Config \times Prog$ où $\Rightarrow$ relie une expression dans une configuration donnée à sa *valeur* et sa configuration finale, et où $\underset{\infty}{\Downarrow}$ dénote l'évaluation infinie d'une expression à partir d'une configuration donnée. Cette évaluation infinie recouvre aussi bien les exécutions productives que divergentes.

Règle d'évaluation des constantes. La règle CONST est une feuille de l'arbre d'évaluation. Elle exprime que toute constante s'évalue en elle-même, sans générer de valeur observable.

$$\textsc{Const}\ \frac{}{conf \vdash c \Downarrow conf \vdash c/\epsilon \qquad c \in \mathcal{V}}$$

Règles d'évaluation du séquencement. Une mise en séquence $A_0; A_1$ peut s'évaluer de trois manières distinctes.

– Si l'évaluation de A_0 puis celle de A_1 termine, alors l'évaluation de $A_0; A_1$ termine également. La trace résultante est la concaténation des deux traces générées par l'évaluation de A_0 et A_1. Elle est nécessairement finie.

$$\textsc{Seq}\ \frac{conf \vdash A_0 \Downarrow conf' \vdash \textbf{skip}/t_0 \qquad conf' \vdash A_1 \Downarrow conf'' \vdash v/t_1}{conf \vdash A_0; A_1 \Downarrow conf'' \vdash v/t_0.t_1}$$

– Si l'évaluation de A_0 est infinie, il est impossible d'évaluer A_1. Le résultat est la trace générée par l'évaluation de A_0.

$$\textsc{Seq-}\infty-0\ \frac{conf \vdash A_0 \underset{\infty}{\Downarrow} /T_0}{conf \vdash A_0; A_1 \underset{\infty}{\Downarrow} /T_0}$$

– Si l'évaluation de A_0 est finie et que l'évaluation de A_1 est infinie, le résultat est la concaténation de la trace finie générée par l'évaluation de A_0 à la trace générée par l'évaluation de A_1.

$$\textsc{Seq-}\infty-1\ \frac{conf \vdash A_0 \Downarrow conf' \vdash \textbf{skip}/t_0 \qquad conf' \vdash A_1 \underset{\infty}{\Downarrow} /T_1}{conf \vdash A_0; A_1 \underset{\infty}{\Downarrow} /t_0.T_1}$$

Règles d'évaluation de la boucle. Dans le cas de la boucle, les divergences peuvent se produire lors de l'évaluation de la condition et lors l'évaluation du corps de la boucle. Il y a donc cinq cas au total. Les deux premiers cas sont ceux des boucles dont l'évaluation termine.

$$\textsc{While-tt}\ \frac{conf \vdash A_0 \Downarrow conf' \vdash \textbf{tt}\ /t_0 \qquad conf' \vdash A_1 \Downarrow conf'' \vdash \textbf{skip}/t_1 \qquad conf'' \vdash \textbf{while}\ A_0\ \textbf{do}\ A_1\ \textbf{done} \Downarrow conf''' \vdash \textbf{skip}/t_2}{conf \vdash \textbf{while}\ A_0\ \textbf{do}\ A_1\ \textbf{done} \Downarrow conf''' \vdash \textbf{skip}/t_0.t_1.t_2}$$

$$\textsc{While-ff}\ \frac{conf \vdash A_0 \Downarrow conf' \vdash \textbf{ff}\ /t_0}{conf \vdash \textbf{while}\ A_0\ \textbf{do}\ A_1\ \textbf{done} \Downarrow conf' \vdash \textbf{skip}/t_0}$$

Les trois autres en cas sont les variantes non-terminantes.

$$\textsc{While-}\infty\textsc{-cond}\ \frac{conf \vdash A_0 \underset{\infty}{\Downarrow} /T_0}{conf \vdash \textbf{while}\ A_0\ \textbf{do}\ A_1\ \textbf{done} \underset{\infty}{\Downarrow} /T_0}$$

$$\textsc{While-}\infty\textsc{-body}\ \frac{conf \vdash A_0 \Downarrow conf' \vdash \textbf{tt}\ /t_0 \qquad conf' \vdash A_1 \underset{\infty}{\Downarrow} /T_1}{conf \vdash \textbf{while}\ A_0\ \textbf{do}\ A_1\ \textbf{done} \underset{\infty}{\Downarrow} /t_0.T_1}$$

$$\textsc{While-}\infty\textsc{-iter}\ \frac{conf \vdash A_0 \Downarrow conf' \vdash \textbf{tt}\ /t_0 \qquad conf' \vdash A_1 \Downarrow conf'' \vdash \textbf{skip}/t_1 \qquad conf'' \vdash \textbf{while}\ A_0\ \textbf{do}\ A_1\ \textbf{done} \underset{\infty}{\Downarrow} /T_2}{conf \vdash \textbf{while}\ A_0\ \textbf{do}\ A_1\ \textbf{done} \underset{\infty}{\Downarrow} /t_0.t_1.T_2}$$

Règles d'évaluation du branchement conditionnel. Le cas du branchement conditionnel est similaire à celui de la boucle. Les évaluations infinies peuvent se produire soit lors de l'évalua-

tion de la condition, soit lors de l'évaluation des branches.

$$\textsc{If-tt}\quad \frac{conf \vdash A_0 \Downarrow conf' \vdash \mathbf{tt}\ /t_0 \qquad conf' \vdash A_1 \Downarrow conf'' \vdash v/t_1}{conf \vdash \mathbf{if}\ A_0\ \mathbf{then}\ A_1\ \mathbf{else}\ A_2 \Downarrow conf'' \vdash v/t_0.t_1}$$

$$\textsc{If-ff}\quad \frac{conf \vdash A_0 \Downarrow conf' \vdash \mathbf{ff}\ /t_0 \qquad conf' \vdash A_2 \Downarrow conf'' \vdash v/t_2}{conf \vdash \mathbf{if}\ A_0\ \mathbf{then}\ A_1\ \mathbf{else}\ A_2 \Downarrow conf'' \vdash v/t_0.t_2}$$

$$\textsc{If-}\infty\textsc{ - cond}\quad \frac{conf \vdash A_0 \underset{\infty}{\Downarrow} /T_0}{conf \vdash \mathbf{if}\ A_0\ \mathbf{then}\ A_1\ \mathbf{else}\ A_2 \underset{\infty}{\Downarrow} /T_0}$$

$$\textsc{If-}\infty\textsc{ - tt}\quad \frac{conf \vdash A_0 \Downarrow conf' \vdash \mathbf{tt}\ /t_0 \qquad conf' \vdash A_1 \underset{\infty}{\Downarrow} /T_1}{conf \vdash \mathbf{if}\ A_0\ \mathbf{then}\ A_1\ \mathbf{else}\ A_2 \underset{\infty}{\Downarrow} /t_0.T_1}$$

$$\textsc{If-}\infty\textsc{ - ff}\quad \frac{conf \vdash A_0 \Downarrow conf' \vdash \mathbf{ff}\ /t_0 \qquad conf' \vdash A_2 \underset{\infty}{\Downarrow} /T_2}{conf \vdash \mathbf{if}\ A_0\ \mathbf{then}\ A_1\ \mathbf{else}\ A_2 \underset{\infty}{\Downarrow} /t_0.T_2}$$

Règles d'évaluation des instructions ayant trait aux variables. Les variables isolées sont évaluées de la même manière que les constantes.

$$\textsc{Var}\quad \frac{}{conf \vdash x \Downarrow conf \vdash x/\epsilon}$$

La construction **new** $x\ : \tau$ **in** A déclare une variable de nom x et de type τ dans l'expression A. La valeur de la variable est initialisée à la valeur par défaut pour le type τ. Le store st est étendu par la variable x pour la seule évaluation de A. Si l'évaluation de A ne termine pas, le résultat est la trace potentiellement infinie générée par A.

$$\textsc{New}\quad \frac{\langle (st \mid x \mapsto default_\tau), X\rangle \vdash A \Downarrow \langle st', X'\rangle \vdash v/t}{\langle st, X\rangle \vdash \mathbf{new}\ x\ : \tau\ \mathbf{in}\ A \Downarrow \langle st' \upharpoonright x, X'\rangle \vdash v/t}$$

$$\textsc{New-}\infty\quad \frac{\langle (st \mid x \mapsto default_\tau), X\rangle \vdash A \underset{\infty}{\Downarrow} /T}{\langle st, X\rangle \vdash \mathbf{new}\ x\ : \tau\ \mathbf{in}\ A \underset{\infty}{\Downarrow} /T}$$

L'assignation $V := A$ évalue A puis V. Il est direct de prouver que le type d'une expression est invariant par réduction. Le système de type nous assure donc que si leurs exécutions terminent,

A a pour résultat une constante v_A de type **exp(**τ**)** et V est une variable x de type **var(**τ**)**. Il est donc cohérent de procéder à la mise à jour de la variable x par la valeur v_A. Dans le cas où l'une ou l'autre de ces évaluations est infinie, l'évaluation de l'expression entière ne termine pas.

$$\textsc{Assign}\quad \frac{conf \vdash A \Downarrow conf' \vdash v_A/t_A \qquad conf' \vdash V \Downarrow \langle st'', X''\rangle \vdash x/t_V}{conf \vdash V := A \Downarrow \langle (st'' \mid x \mapsto v_A), X''\rangle \vdash \textbf{skip}/t_A.t_V}$$

$$\textsc{Assign-}\infty - rhs\quad \frac{conf \vdash A \underset{\infty}{\Downarrow} /T_A}{conf \vdash V := A \underset{\infty}{\Downarrow} /T_A}$$

$$\textsc{Assign-}\infty - lhs\quad \frac{conf \vdash A \Downarrow conf' \vdash v_A/t_A \qquad conf' \vdash V \underset{\infty}{\Downarrow} /T_V}{conf \vdash V := A \underset{\infty}{\Downarrow} /t_A.T_V}$$

Le déréférencement $!V$ procède à l'évaluation d'une expression V en une variable, et à l'accès à la valeur de cette variable à travers le store.

$$\textsc{Deref}\quad \frac{conf \vdash V \Downarrow \langle st', X'\rangle \vdash x/t_V}{conf \vdash !V \Downarrow \langle st', X'\rangle \vdash st'(x)/t_V}$$

$$\textsc{Deref-}\infty\quad \frac{conf \vdash V \underset{\infty}{\Downarrow} /T_V}{conf \vdash !V \underset{\infty}{\Downarrow} /T_V}$$

Règles d'évaluation des instructions synchrones. Les règles précédentes sont classiques. La particularité de PsyALGOL tient dans ses instructions synchrones à commencer par l'instruction **advance** qui est la seule à générer les traces. Afin de justifier nos définitions plus avant, nous donnons quelques détails sur son fonctionnement dans le langage PsyC. Ce langage définit les variables temporelles comme un sous-ensemble particulier des variables du programme, utilisables comme telles. L'état observable d'un agent est constitué de la séquence des valeurs de ces variables temporelles, échantillonnées à chaque **advance**. Il est donc équivalent de donner explicitement à chaque instruction **advance** un vecteur de valeurs à rendre observable. C'est précisément l'approche que nous avons adoptée dans PsyALGOL. L'état observable d'un programme PsyC comprend également les messages émis. Les dates de visibilité de ses messages ne coïncident pas avec les instructions **advance**, mais restent *relatives* à ces dernières. Nous prenons ce fait en compte dans notre sémantique opérationnelle.

Une instruction **advance**$(n,\ E_1, \ldots, E_O)$ procède à l'évaluation successive des expressions E_i, dont les valeurs sont rassemblées sous la forme d'un vecteur $\overrightarrow{outvts} = (v_1, \ldots, v_O)$ pour former une partie du nouvel état observable du programme. Les messages visibles avant l'**advance**

en cours d'évaluation sont ajoutés à la trace de sortie. La règle correspondante est définie ainsi :

ADVANCE

$$\frac{\begin{array}{c} conf_0 \vdash E_1 \Downarrow conf_1 \vdash v_1/t_1 \\ \vdots \\ conf_{O-1} \vdash E_O \Downarrow \langle st_O, cx_O, inbox_O, \overrightarrow{outvts_O}, outbox_O \rangle \vdash v_O/t_O \end{array}}{conf_0 \vdash \textbf{advance}(n,\ E_1, \ldots, E_O) \Downarrow final \vdash \textbf{skip}/t_1 \ldots t_O.t_{adv}}$$

Commençons par définir la trace t_{adv} générée par l'**advance**. Cette trace est constituée par la valeur des variables temporelles $\overrightarrow{outvts_O}$ et par les messages de $outbox_O$ devenant visibles avant l'**advance** considéré. Nous commençons par générer la trace dépourvue de messages :

$$\begin{array}{rcl} t_{vt} & = & \overrightarrow{outvts_O}^n \\ & = & \overrightarrow{outvts_O}.\overrightarrow{outvts_O} \ldots \overrightarrow{outvts_O}\ n \text{ fois} \end{array}$$

La séquence des messages de $outbox_O$ devenant visibles est la restriction de $outbox_O$ aux messages dont la date de visibilité est inférieure à n :

$$\begin{array}{rcl} Sent & = & \mathsf{filter}\,((m,t) \mapsto t \leq n,\ outbox_O) \\ & = & (m_1,t_1).(m_2,t_2) \ldots (m_k,t_k) \end{array}$$

Ces messages doivent être rajoutés à t_{vt} afin de former la trace finale t_{adv}. Nous définissons une fonction intermédiaire update permettant d'ajouter un message (m,t) à une trace :

$$\begin{array}{l} \mathsf{update} : \Sigma_{msg} \times \mathbb{Z} \times \Sigma_{tr} \rightharpoonup \Sigma_{tr} \\ \mathsf{update}(m,t,tr)[i] \qquad \textit{indéfinie si } i \geq |tr| \\ \mathsf{update}(m,t,tr)[i] \quad = \quad tr[i] \textit{ si } i \neq t \textit{ et } i < |tr| \\ \mathsf{update}(m,t,tr)[t-1] \quad = \quad (\overrightarrow{vt}, m.s) \textit{ où } (\overrightarrow{vt}, s) \ = \ tr[t-1] \end{array}$$

La trace finale t_{adv} résulte de l'application successive de cette fonction aux messages de $Sent$ à partir de la trace t_{vt} :

$$t_{adv} = \mathsf{update}(m_k, t_k, \mathsf{update}(m_{k-1}, t_{k-1}, \ \ldots \mathsf{update}(m_2, t_2, \mathsf{update}(m_1, t_1, t_{vt}))\ \ldots)).$$

Il nous reste à calculer la configuration finale $final$. Commençons par le nouvel état de l'environnement :

$$(cx_f, t_{in}) = \mathsf{next}_{tr}(t_{adv}, cx_O, \epsilon).$$

Par définition de next_{tr}, cx_f est le nouvel état de l'environnement, et $input$ est la "trace de sortie" de ce même environnement calculé à partir de cx_O en lisant t_{adv}. Par symétrie, t_{in} est également une portion du flot d'entrée du programme en cours d'exécution. Extrayons de ce flot les messages envoyés au programme :

$$\begin{array}{rcl} Received & = & \mathsf{messages}(t_{in}) \\ & = & r_1.r_2 \ldots r_l. \end{array}$$

Ces messages sont concaténés à l'ancienne séquence des messages en entrée :

$$inbox_f = inbox_O.r_1.r_2 \ldots r_l.$$

Pour les messages en sortie, la nouvelle file d'attente est calculée de façon complémentaire à $Sent$:

$$outbox_f \quad = \quad \mathsf{filter}\,((m,t) \mapsto t > n,\ outbox_O)\,.$$

Nous définissons la configuration finale $final$ comme l'agrégation de ces résultats :

$$final = \langle st_O, cx_f, inbox_f, \overrightarrow{outvts}, outbox_f \rangle.$$

Chaque composante est définie ci-dessous :

- st_O est le store final après évaluation de l'expression E_O ;
- cx est l'environnement extérieur au programme ;
- $inbox_f$ est la file d'attente des messages en entrée dans le nouvel état du programme ;
- $\overrightarrow{outvts_{ok}} = (v_1, \ldots, v_O)$ est le vecteur de valeurs issues de l'évaluation de $E_1 \ldots E_n$;
- $outbox_f$ est la file d'attente des messages en sortie.

Dans le cas où l'évaluation d'une des expressions E_i est infinie, l'instruction **advance** n'est pas calculée. Cela donne à la règle ADVANCE la forme suivante.

ADVANCE-∞-J

$$\frac{conf \vdash E_1 \Downarrow conf_1 \vdash v_1/t_1 \qquad \ldots \qquad conf_{j-1} \vdash E_j \underset{\infty}{\Downarrow} /T_j \qquad j \leq O}{conf \vdash \mathbf{advance}(n,\ E_1, \ldots, E_O) \underset{\infty}{\Downarrow} /t_1.\ldots.t_{j-1}.T_j \qquad n > 0}$$

La sémantique opérationnelle de quelques autres instructions reste à définir. Ces instructions permettent d'accéder au contenus des files d'attentes de messages ainsi qu'aux valeurs des variables temporelles en entrée. Cette dernière opération est réalisée par l'instruction **get**$_i$, dont le comportement est défini dans la règle INPUT. Cette règle correspond simplement à la lecture de l'état observable du contexte.

INPUT

$$\frac{}{\langle st, cx, X\rangle \vdash \mathbf{get}_i \Downarrow \langle st, syst, X\rangle \vdash pi_i\ (pi_1(\mathsf{input}(cx)))/\epsilon \qquad i \in [1; \mathcal{I}(vt)]}$$

Deux instructions sont utilisées pour respectivement envoyer et lire les messages : **send**$_i(A, n)$ et **take**$_i$. Afin d'envoyer un message, il est nécessaire de préciser la boite à lettre de sortie ainsi que le délai avant l'apparition du message.

SEND

$$\frac{conf \vdash A \Downarrow \langle X', outbox'\rangle \vdash v/t}{conf \vdash \mathbf{send}_i(A, n) \Downarrow \langle X', \mathsf{insert}(v, n, outbox')\rangle \vdash \mathbf{skip}/t \qquad i \in [1; \mathcal{O}(msg)]}$$

Dans le cas où l'évaluation de A ne termine pas, l'envoi de message n'est pas effectué.

SEND-∞

$$\frac{conf \vdash A \underset{\infty}{\Downarrow} /T_A}{conf \vdash \mathbf{send}_i(A, n) \underset{\infty}{\Downarrow} /T_A \qquad i \in [1; \mathcal{O}(msg)]}$$

Contrairement aux variables temporelles, les messages ne sont accessibles qu'une seule fois.

TAKE

$$\frac{}{\langle X, inbox, Y\rangle \vdash \mathbf{take}_i \Downarrow \langle X, inbox', Y\rangle \vdash v/\epsilon \qquad i \in [1; \mathcal{I}(msg)]} \quad \mathsf{extract}(inbox) = (v, inbox')$$

Enfin, il est nécessaire de pouvoir accéder au nombre courant de messages en attente.

TAKE

$$\frac{}{\langle X, inbox, Y\rangle \vdash \mathbf{count}_i \Downarrow \langle X, inbox, Y\rangle \vdash |inbox|/\epsilon \qquad i \in [1; \mathcal{I}(msg)]}$$

A.3 Remarques de conclusion

La complexité de la sémantique opérationnelle donnée plus haut est caractéristique d'un langage pensé pour une utilisation dans un cadre industriel. Les primitives d'envoi et de réception de messages mêlées aux traits synchrones rendent la formalisation relativement ardue. Qui plus est, notre formalisation ignore certains aspects du fonctionnement de PsyC.

- Il est possible de prendre plus de messages qu'il n'y en a dans la file de réception. En PsyC, l'extraction des messages n'est possible que via une construction ad-hoc du langage, empêchant ce comportement erroné.
- Les variables temporelles sont dotées d'un historique permettant d'accéder à leurs valeurs passées sur une profondeur finie. Ceci est encodable en PsyALGOL.
- Des restrictions syntaxiques sur les bornes des boucles permettent de s'assurer que la consommation mémoire (en particulier pour les boites aux lettres) est bornée et statiquement calculable.

Afin de faire le lien avec les développements de ce manuscrit, nous pouvons remarquer que la sémantique opérationnelle de PsyALGOL procède à une décomposition des sauts temporels provoqués par les instructions **advance**$(n, \_)$ en générant pour ces instructions des traces de longueur n et en consommant autant de données en entrée. Les échanges de données entre le programme et son contexte (c'est-à-dire les autres programmes) sont effectués sur cette base temporelle, de façon synchrone. La nature synchrone faible est montrée par la règle **advance**, où la trace de sortie est générée *avant* la phase de communication avec le contexte. Un programme PsyALGOL calcule une fonction déterministe sur des séquences de données potentiellement infinies et est implémentable par une machine de Moore (à états finis si l'arbre de dérivation admet une représentation finie).